Е. Е. Бизянов, А. А. Гутник

ПРОГРАММИРОВАНИЕ

Лабораторный практикум

Лабораторный практикум

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
ЛУГАНСКОЙ НАРОДНОЙ РЕСПУБЛИКИ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«ДОНБАССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
КАФЕДРА СПЕЦИАЛИЗИРОВАННЫХ КОМПЬЮТЕРНЫХ СИСТЕМ

Е. Е. Бизянов, А. А. Гутник

ПРОГРАММИРОВАНИЕ

Лабораторный практикум

Рекомендовано Ученым советом
факультета автоматизации и электротехнических систем

Алчевск
ГОУ ВПО ЛНР «ДонГТУ»
2018

УДК 004.45
ББК 32.973-18.02
Б59

Рецензент

А. Н. Баранов — к.т.н., доц. кафедры специализированных компьютерных систем ГОУ ВПО ЛНР «ДонГТУ».

Рекомендовано Ученым советом
факультета автоматизации и электротехнических систем
(Протокол № 10 от 18.06.2018)

Бизянов Е. Е.

Б59 Программирование : лаб. практикум / Е. Е. Бизянов, А. А. Гутник. — Алчевск : ГОУ ВПО ЛНР «ДонГТУ», 2018. — 191 с.

Практикум предназначен для приобретения практических навыков по созданию программ на языке программирования Java, а также основным приемам, принятым в практическом программировании. Пособие построено в виде лабораторного практикума, содержащего теоретические сведения по каждой теме, контрольные вопросы, задания и методические указания к его выполнению.

Для студентов 2 курса направления подготовки 09.03.01 «Информатика и вычислительная техника» всех форм обучения.

УДК 004.45
ББК 32.973-18.02

© ГОУ ВПО ЛНР «ДонГТУ», 2018
© Е. Е. Бизянов, А. А. Гутник 2018
© Н. В. Чернышова, художественное оформление обложки, 2018

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В лабораторном практикуме изложены теоретические положения разработки программного обеспечения с использованием технологии объектно-ориентированного программирования. Подробно рассмотрены основные приемы решения задач различных классов.

Рассмотрение возможностей системы программирования и конструкций языка программирования позволяет студентам с первых занятий получить представление о технологии разработки приложений. По окончании курса студент должен приобрести все необходимые навыки и умения для проектирования графических приложений высокой сложности.

В качестве языка программирования используется язык Java. В практикуме представлено 5 лабораторных работ, приведены теоретические сведения, необходимые для выполнения лабораторных работ по курсу «Программирование», а также вопросы для самоконтроля.

Содержание практикума соответствует программе дисциплины «Программирование» для студентов 2 курса направления подготовки 09.03.01 «Информатика и вычислительная техника».

Материал пособия может быть полностью или частично использован преподавателем для организации лабораторных работ в соответствии с отведенным объемом часов.

1 ЛАБОРАТОРНАЯ РАБОТА №1

Тема работы: разработка визуальных приложений Java.

Цель работы: изучить классы менеджеров компоновки пакета Java AWT.

1.1 Принципы построения графического интерфейса. Базовые классы

В Java при разработке библиотеки Abstraction Window Toolkit (AWT) были созданы абстрактные классы, подходящие для библиотек каждой из операционных систем. Таким образом, графическое приложение, написанное на Java, выглядит в любой из операционных систем схожим образом.

В AWT компонентом считается объект класса `Component` или объект всякого класса, расширяющего класс `Component`.

`Component` — это абстрактный класс, который инкапсулирует все атрибуты визуального интерфейса — обработку ввода с клавиатуры, управление фокусом, взаимодействие с мышью, уведомление о входе/выходе из окна, изменение размеров и положения окон, прорисовку собственного графического представления, сохранение текущего текстового шрифта, цветов фона и переднего плана (более 100 методов).

Каждый компонент перед выводом на экран помещается в контейнер. Поэтому, создав объект класса `Component` или его наследника, следует добавить его к предварительно созданному объекту класса `Container` или его наследника одним из методов `add()`.

`Container` — это абстрактный подкласс класса `Component`, определяющий дополнительные методы, дающие возможность помещать в него другие компоненты, обеспечивая таким образом построение иерархической системы визуальных объектов.

`Container` отвечает за расположение содержащихся в нем компонентов с помощью интерфейса `LayoutManager`. В контейнер наряду с компонентами можно помещать другие контейнеры, в которых находятся иные компоненты, достигая тем самым большей гибкости в расположении компонентов.

1.2 Графическое приложение Java

Основное окно приложения, взаимодействующее с операционной системой, необходимо строить по правилам графической системы. Такое окно в готовом виде описано в классе `Frame`. Чтобы создать окно, достаточно сделать создаваемый класс потомком класса `Frame`, как показано ниже:

```
import java.awt.*;
class TooSimpleFrame extends Frame{
  public static void main(String[] args){
    Frame fr = new TooSimpleFrame();
  fr.setSize(400, 150);
  fr.setVisible(true);
  }
}
```

Результат работы программы приведен на рисунке 1.1.

Рисунок 1.1 — Приложение без строки заголовка

Класс `TooSimpleFrame` обладает всеми свойствами класса `Frame`, являясь его наследником. В нем создается экземпляр окна `fr`, и методом `setSize()` устанавливаются его размеры — 400x150 пикселей. Если не задать размер окна, то на экране оно будет иметь минимальный размер — только строку заголовка. Окно выводится на экран методом `setVisible(true)`. С точки зрения библиотеки AWT, чтобы создать окно, следует выделить область оперативной памяти, заполненную нужными графическими данными, и отобразить содержимое этой области на экране методом `setVisible(true)`.

Такое приложение нельзя завершить стандартными средствами. Необходимо завершать работу приложения средствами операционной системы, например, комбинацией клавиш <Ctrl>+<C>.

Добавим заголовок окна и обращение к методу, позволяющему завершить приложение, пример исходного кода представлен ниже:

```
import java.awt.*;
import java.awt.event.*;
class SimpleFrame extends Frame{
  SimpleFrame(String s){
    super (s);
    setSize(400, 150);
    setVisible(true);
    addWindowListener(new WindowAdapter(){
      public void windowClosing(WindowEvent ev)
        { System.exit(0);  }
    });
  }
  public static void main(String[] args){
    new SimpleFrame("Моя программа");
  }
}
```

Результат работы программы приведен на рисунке 1.2.

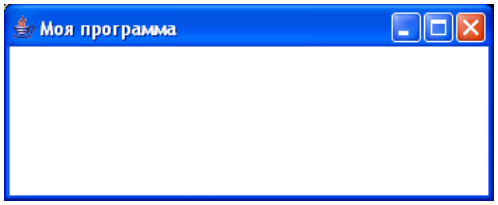


Рисунок 1.2 — Приложение со строкой заголовка

В программу добавлен конструктор класса `SimpleFrame`, обращающийся к конструктору суперкласса `Frame`, который записывает аргумент `s` в строку заголовка окна.

В конструктор помещены методы установки размеров окна, вывода его на экран и добавлено обращение к методу `addWindowListener()`, реагирующему на действия с окном. В качестве аргумента в этот метод передается экземпляр безымянного внутреннего класса, расширяющего класс `WindowAdapter`. Этот безымянный класс реализует метод `WindowClosing()`,

обрабатывающий попытку закрытия окна. Реализация метода достаточно проста — приложение завершается методом `exit()` класса `System`.

Ниже приведен код программы, выводящей надпись «Hello, XXI century World!», результат работы которой представлен на рисунке 1.3:

```
import java.awt.*;
import java.awt.event.*;
class HelloWorld extends Frame {
  HelloWorld(String s) {
    super(s);
  }
  public void paint(Graphics g) {
    g.setFont(new Font("Serif", Font.ITALIC |
Font.BOLD, 30));
    g.drawString("Hello, XXI century World!", 20,
100);
  }
  public static void main(String[] args) {
    Frame f = new HelloWorld("Здравствуй, мир XXI
века!");
    f.setSize(400, 150);
    f.setVisible(true);
    f.addWindowListener(new WindowAdapter() {
     public void windowClosing(WindowEvent ev) {
        System.exit(0);
     }
    });
  }
}
```

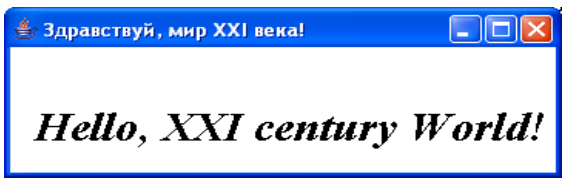


Рисунок 1.3 — Графическая программа с приветствием

Для вывода текста используется метод `paint()` класса `Component`, унаследованный классом `Frame` пустым, т.е. мы сами должны написать его.

Метод `paint()` принимает в качестве аргумента экземпляр объекта `g` класса `Graphics`, который выводит текст методом `drawString()`. Мы также должны указать положение начала строки в окне — 20 пикселей от левого края и 100 пикселей сверху. Эта точка — левая нижняя точка первой буквы текста H.

Кроме того, для надписи установим шрифт «Serif» размером 30 пунктов, полужирный, курсивный. Любой шрифт — объект класса `Font`, и устанавливается он методом `setFont()` класса `Graphics`.

Вызовы методов установки размеров окна, вывода его на экран и завершения программы вынесены в метод `main()`.

1.3 Контейнеры

1.3.1 Класс Component

Класс `Component` — основа библиотеки AWT — обладает большими возможностями. В нем заданы пять статических констант, определяющих размещение компонента внутри пространства, выделенного него в контейнере: `BOTTOM_ALIGNMENT`, `CENTER_ALIGNMENT`, `LEFT_ALIGNMENT`, `RIGHT_ALIGNMENT`, `TOP_ALIGNMENT` и около сотни методов.

Большинство методов — это методы доступа вида: `getxxx()`, `isxxx()`, `setxxx()`.

Конструктор данного класса нам недоступен (он объявлен с модификатором доступа — `protected`). Класс `Component` абстрактный, поэтому мы не можем создавать объекты этого класса в программе, но может создавать его наследников.

Компонент всегда занимает прямоугольную область со сторонами, параллельными сторонам контейнера, и имеет определенные размеры, измеряемые в пикселях, которые можно узнать методом `getSize()`, возвращающим объект класса `Dimension`, или целочисленными методами `getHeight()` и `getWidth()`, возвращающими высоту и ширину прямоугольника. Новый размер компонента можно установить методами `setSize(Dimension d)`

или `setSize(int width, int height)`, если это допускает менеджер компоновки контейнера, содержащего компонент.

У компонента есть предпочтительный размер, при котором компонент выглядит пропорционально. Этот размер можно прочитать методом `getPreferredSize()` в виде объекта `Dimension`.

Компонент обладает минимальным и максимальным размерами. Их возвращают методы `getMinimumSize()` и `getMaximumSize()` в виде объекта `Dimension`.

У компонента есть своя система координат. Ее начало — точка с координатами (0, 0) — находится в левом верхнем углу компонента, ось `x` идет вправо, ось `y` — вниз, а координатные точки расположены между пикселями.

Координаты левого верхнего угла компонента в системе координат контейнера можно узнать методами `getLocation()`, а также изменить методами `setLocation()` (если это позволяет менеджер компоновки).

Можно выяснить сразу и положение, и размер прямоугольной области компонента методом `getBounds()`, возвращающим объект класса `Rectangle`, и изменить положение и размер компонента методами `setBounds()`, если это позволяет сделать менеджер компоновки.

Доступность компонента для действий пользователя можно проверить методом `isEnabled()`, а изменить — методом `setEnabled(boolean enable)`.

Для многих компонентов доступен графический контекст — объект класса `Graphics`, — который можно получить методом `getGraphics()` и управляется методом `paint()`.

У контекста компонента есть цвет переднего плана и цвет фона — объекты класса `Color`. Цвет фона можно получить методом `getBackground()`, а изменить — методом `setBackground(Color color)`. Цвет переднего плана можно получить методом `getForeground()`, а изменить — методом `setForeground(Color color)`.

Для компонента определен курсор, показывающий положение мыши, — объект класса `Cursor`. Сведения о курсоре можно получить методом `getCursor()`.

1.3.2 Класс Container

Класс `Container` — прямой наследник класса `Component`, и поэтому он наследует все его методы. Основу класса составляют методы добавления компонентов в контейнер:

– `add (Component comp)` — компонент `comp` добавляется в конец контейнера;

– `add (Component comp, int index)` — компонент `comp` добавляется в позицию `index` в контейнере, если `index=-1`, то компонент добавляется в конец контейнера;

– `add (Component comp, object constraints)` — компонент `comp` добавляется в контейнер, менеджеру компоновки контейнера даются указания объектом `Constraints`;

– `add (String name, Component comp)` — компонент `comp` добавляется в контейнер и получает имя `name`.

Два метода удаляют компоненты из контейнера:

– `remove (Component comp)` — удаляет компонент с именем `comp`;

– `remove (int index)` — удаляет компонент с индексом `index` в контейнере.

Один из компонентов в контейнере обычно получает фокус ввода, и на него направляется ввод с клавиатуры. Фокус можно переносить с одного компонента на другой клавишами <Tab> и <Shift>+<Tab>. Компонент может запросить фокус методом `requestFocus()` и передать фокус следующему компоненту методом `transferFocus()`. Компонент может проверить, имеет ли он фокус, своим логическим методом `hasFocus()`. Эти методы принадлежат классу `Component`.

Для размещения компонентов в контейнере необходимо определить менеджер компоновки `(layout manager)` — объект, реализующий интерфейс `LayoutManager` или его интерфейс-наследник `LayoutManager2`.

В контейнере в любой момент времени может быть задан только один менеджер компоновки. В каждом контейнере есть свой менеджер по умолчанию, установка другого менеджера производится методом `setLayout(LayoutManager manager)`.

Для компоновки компонентов «вручную» можно отключить менеджер по умолчанию методом `setLayout(null)`.

Кроме событий класса `Component`: `ComponentEvent`, `FocusEvent`, `KeyEvent`, `MouseEvent` при добавлении и удалении компонентов в контейнере происходит событие `ContainerEvent`.

1.3.3 Контейнер Panel

Контейнер `Panel` — это невидимый компонент графического интерфейса, служащий для объединения нескольких других компонентов в одну группу.

В классе `Panel` определены два конструктора:

- `Panel()` — создает контейнер с менеджером компоновки по умолчанию `FlowLayout`;
- `Panel(LayoutManager layout)` — создает контейнер с указанным менеджером компоновки `layout`.

После создания контейнера в него следует добавить компоненты унаследованным методом `add()`:

```
Panel p = new Panel();
p.add(compl);
p.add(comp2);
```

и т. д. Размещает компоненты в контейнере его менеджер компоновки.

1.3.4 Контейнер ScrollPane

Контейнер `ScrollPane` может содержать только один компонент, который не помещается целиком в окне, обеспечивая средства прокрутки для просмотра большого компонента, — можно установить полосы прокрутки постоянно видимыми — константой `SCROLLBARS_ALWAYS`, либо сделать так, чтобы они появлялись только

при необходимости (если компонент действительно не помещается в окно) константой SCROLLBARS_AS_NEEDED.

Если полосы прокрутки не установлены (константа имеет значение SCROLLBARS_NEVER), то перемещение компонента для просмотра нужно обеспечить из программы одним из методов setScrollPosition().

В классе два конструктора:

– ScrollPane() — создает контейнер, в котором полосы прокрутки появляются по необходимости;

– ScrollPane(int scrollbars) — создает контейнер, в котором появление линеек прокрутки задается одной из трех указанных выше констант.

Конструкторы создают контейнер размером 100x100 пикселей, но при необходимости можно изменить размер методом setSize(int width, int height).

Ограничение, заключающееся в том, что scrollPane может содержать только один компонент, можно обойти, сделав этим единственным компонентом объект класса Panel, и разместив на панели группу компонентов.

Методы, позволяющие «прокручивать» компонент в ScrollPane:

– методы getHAdjustable() и getVAdjustable() возвращают положение линеек прокрутки в виде интерфейса Adjustable;

– метод getScrollPosition() показывает координаты (x, y) точки компонента, находящейся в левом верхнем углу панели ScrollPane, в виде объекта класса Point;

– метод setScrollPosition(Point p) или setScrollPosition(int x, int y) прокручивает компонент в позицию (x, y).

1.3.5 Контейнер Window

Контейнер Window — это пустое окно, без внутренних элементов: рамки, строки заголовка, строки меню, полос прокрутки.

Окно типа window не надо заносить в контейнер методом add(), так как оно не связано с оконным менеджером графической системы. Следовательно, нельзя изменить его размеры, переместить в другое место экрана. Поэтому оно может быть создано только каким-нибудь уже существующим окном — владельцем (owner) или родителем (parent) окна window. Когда окно-владелец удаляется с экрана, вместе с ним удаляется и порожденное окно. Владелец окна указывается в конструкторе:

– window (Frame f) — создает окно, владелец которого — фрейм f;

– window (window owner) — создает окно, владелец которого уже имеющееся окно или подкласс класса window.

Созданное конструктором окно следует отобразить методом show(). Убрать окно с экрана можно методом hide(), а проверить, видно ли окно на экране — логическим методом isShowing().

Окно типа window возможно использовать для создания всплывающих окон предупреждения, сообщения, подсказок. Видимое на экране окно выводится на передний план методом toFront(), помещается на задний план методом toBack(). Уничтожить окно, освободив занимаемые им ресурсы, можно методом dispose().

Менеджер компоновки в окне по умолчанию – BorderLayout. Окно создает свой экземпляр класса Toolkit, который можно получить методом getToolkit().

Кроме событий класса Component: ComponentEvent, FocusEvent, KeyEvent, MouseEvent, при изменении размеров окна, его перемещении или удалении с экрана, а также вывода на экран происходит событие WindowEvent.

1.3.6 Контейнер Frame

Контейнер Frame — это полноценное окно со строкой заголовка, в которой размещены кнопки контекстного меню, сворачивания окна в ярлык и разворачивания во весь экран и кнопка закрытия. Заголовок окна передается как параметр в конструктор, или задается методом setTitle(string title). Окно окружено рамкой, в него также

можно устанавливать меню методом `setMenuBar (MenuBar mb)`. Метод `setIconImage(image icon)` — помещает изображение в строку заголовка окна.

Все элементы окна `Frame` вычерчиваются графической оболочкой операционной системы по ее правилам. Окно типа `Frame` автоматически регистрируется в оконном менеджере графической оболочки и может перемещаться, менять размеры, сворачиваться в панель задач с помощью мыши или клавиатуры.

Создать окно типа `Frame` можно следующими конструкторами:

– `Frame()` — создает окно с пустой строкой заголовка;

– `Frame(string title)` — записывает строку `title` в заголовок.

Класс `Frame` наследует около двухсот методов классов `Component`, `Container` и `Window`. Менеджер компоновки по умолчанию — `BorderLayout`. Методы класса `Frame` осуществляют доступ к элементам окна.

Кроме событий класса `Component`: `ComponentEvent`, `FocusEvent`, `KeyEvent`, `MouseEvent`, при изменении размеров окна, его перемещении или удалении с экрана, а также вывода на экран происходит событие `WindowEvent`.

1.3.7 Контейнер Dialog

Контейнер `Dialog` — это окно фиксированного размера, предназначенное для ответа на сообщения приложения. Окно диалога автоматически регистрируется в оконном менеджере графической оболочки, следовательно, его можно перемещать по экрану, и менять его размеры. Но окно типа `Dialog`, как и его суперкласс — окно типа `Window`, — обязательно имеет владельца `owner`, который указывается в конструкторе. Окно типа `Dialog` может быть модальным (`modal`), в котором надо обязательно выполнить все предписанные действия, иначе из окна нельзя будет выйти.

В классе семь конструкторов:

– `Dialog (Dialog owner)` — создает немодальное диалоговое окно с пустой строкой заголовка;

– Dialog (Dialog owner, string title) — создает немодальное диалоговое окно со строкой заголовка title;

– Dialog(Dialog owner, String title, boolean modal) — создает диалоговое окно, которое будет модальным, если modal=true.

Четыре других конструктора аналогичны, но создают диалоговые окна, принадлежащие окну типа Frame:

– Dialog(Frame owner);

– Dialog(Frame owner, String title);

– Dialog(Frame owner, boolean modal);

– Dialog(Frame owner, String title, Boolean modal).

Для контейнера определены методы: isModal(), проверяющий состояние модальности, и setModal(boolean modal), меняющий это состояние.

Кроме событий класса Component: ComponentEvent, FocusEvent, KeyEvent, MouseEvent, при изменении размеров окна, его перемещении или удалении с экрана, а также вывода на экран происходит событие WindowEvent.

1.3.8 Контейнер FileDialog

Контейнер FileDialog — это модальное окно с владельцем типа Frame, содержащее стандартное окно выбора файла операционной системы для открытия (константа LOAD) или сохранения (константа SAVE). Окна операционной системы создаются и помещаются в объект класса FileDialog автоматически.

В классе три конструктора:

– FileDialog (Frame owner) — создает окно с пустым заголовком для открытия файла;

– FileDialog (Frame owner, String title) — создает окно открытия файла с заголовком title;

– FileDialog(Frame owner, String title, int mode) — создает окно открытия или сохранения документа; аргумент

mode может иметь значения: FileDialog.LOAD и FileDialog.SAVE.

Методы класса getDirectory() и getFile() возвращают только выбранный каталог и имя файла в виде строки String. Загрузку или сохранение файла затем нужно производить методами классов ввода/вывода.

Можно установить начальный каталог для поиска файла и его имя методами setDirectory(String dir) и setFile(String fileName). Вместо конкретного имени файла fileName можно указать некоторый шаблон, например, *.java (первые символы — звездочка и точка), тогда в окне будут видны только имена файлов, заканчивающиеся точкой и расширением java.

Кроме событий класса Component: ComponentEvent, FocusEvent, KeyEvent, MouseEvent, при изменении размеров окна, его перемещении или удалении с экрана, а также вывода на экран происходит событие WindowEvent.

1.4 Размещение компонентов

1.4.1 Менеджер FlowLayout

Менеджер компоновки FlowLayout помещает в контейнер один компонент за другим слева направо, переходя от верхних рядов к нижним, наподобие того, как появляются символы в окне редактора текста. Компоненты размещаются в том порядке, в каком они заданы в методах add().

Конструкторы для создания экземпляра FlowLayout:

– без указания выравнивания и зазора между компонентами — public FlowLayout();

– с указанием выравнивания — public FlowLayout(int align);

– с указанием выравнивания и зазора между компонентами по вертикали и горизонтали — public FlowLayout(int align, int hgap, int vgap).

Align — выравнивание элементов управления, задается одной из констант: CENTER (по центру), LEFT (по левому краю), RIGHT (по

правому краю); hgap и vgap — горизонтальные и вертикальные промежутки между компонентами, задаются в пикселях.

Первый конструктор класса FlowLayout не имеет параметров. Он устанавливает по умолчанию режим центрирования компонентов и зазор между компонентами по вертикали и горизонтали, равный 5 пикселям.

Второй конструктор позволяет выбрать режим компоновки с заданным выравниванием компонентов в окне контейнера по горизонтали. В качестве параметров этому конструктору необходимо передавать значения FlowLayout.LEFT, FlowLayout.RIGHT, или FlowLayout.CENTER. Зазор между компонентами будет при этом равен по умолчанию 5 пикселям.

Третий конструктор допускает раздельное указание режима выравнивания, а также зазоров между компонентами по вертикали и горизонтали в пикселях.

Метод add(Component) помещает компонент в контейнер.

Ниже приведен исходный код с размещением пяти кнопок на форме с помощью менеджера FlowLayout. Результат работы программы показан на рисунке 1.4.

```
import java.awt.*;
import java.awt.event.*;
class FlowTest extends Frame{
  FlowTest(String s) {
   super(s);
   setLayout(new   FlowLayout(FlowLayout.LEFT,   10,
10));
   add(new Button("Кнопка 1"));
   add(new Button("Кнопка 2"));
   add(new Button("Кнопка 3"));
   add(new Button("Кнопка 4"));
   add(new Button("Кнопка 5"));
   setSize(300, 100);
   setVisible(true);
  }
  public static void main(String[] args) {
```

```
    Frame f = new FlowTest("Менеджер FlowLayout");
    f.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent ev)
      {
        System.exit(0);
      }
    });
  }
}
```

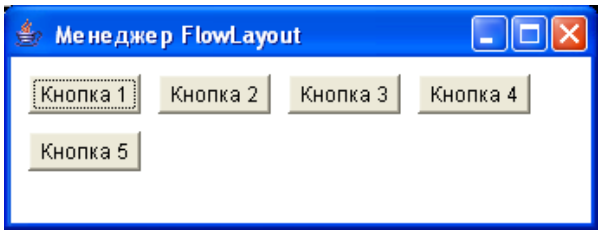


Рисунок 1.4 — Размещение компонентов с помощью FlowLayout

Компонент `Button` — это кнопка стандартного для данной графической системы вида. Конструктор `Button (string label)` создают кнопку с надписью `label`.

1.4.2 Менеджер BorderLayout

Менеджер компоновки `BorderLayout` делит контейнер на пять неравных областей, полностью заполняя каждую область одним компонентом. Области получили географические названия `NORTH` (север), `SOUTH` (юг), `WEST` (запад), `EAST` (восток) и `CENTER` (центр).

Метод `add()` для `BorderLayout` принимает два аргумента: ссылку на компонент `comp` и область `region`, в которую помещается компонент в область одной из перечисленных выше констант: `add(Component comp, String region)`. Обычный метод `add (Component comp)` с одним аргументом помещает компонент в область `CENTER`.

В классе определены два конструктора:

– `BorderLayout()` — между областями нет промежутков;

– BorderLayout(int hgap, int vgap) — между областями остаются горизонтальные hgap и вертикальные vgap промежутки, задаваемые в пикселях.

Если в контейнер помещается менее пяти компонентов, то некоторые области не используются и не занимают места в контейнере. Если не занята область CENTER, то компоненты прижимаются к границам контейнера.

Ниже представлен исходный код приложения с пятью кнопками, размещенными с помощью менеджера BorderLayout. Результат работы программы показан на рисунке 1.5.

```
import java.awt.*;
import java.awt.event.*;
class BorderTest extends Frame{
BorderTest(String s) {
    super(s);
    add(new            Button("Кнопка            1"),
BorderLayout.NORTH);
    add(new            Button("Кнопка            2"),
BorderLayout.SOUTH);
    add(new Button("Кнопка 3"), BorderLayout.WEST);
    add(new Button("Кнопка 4"), BorderLayout.EAST);
    add(new            Button("Кнопка            5"),
BorderLayout.CENTER);
    setSize(300, 100);
    setVisible(true);
}
public static void main(String[] args) {
    Frame    f    =    new    BorderTest("Менеджер
BorderLayout");
    f.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent ev)
    {
        System.exit(0);
    }
});
```

```
  }
}
```

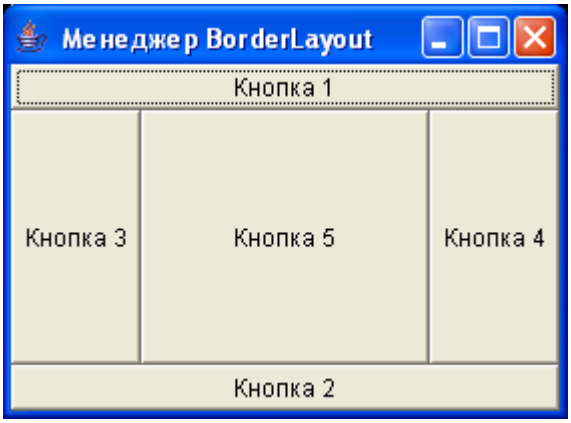


Рисунок 1.5 — Области компоновки BorderLayout

Обычно в каждую область менеджера компоновки `BorderLayout` помещают не один компонент, а панель с группой компонентов на ней.

1.4.3 Менеджер GridLayout

При использовании менеджера компоновки `GridLayout` компоненты размещаются в ячейках виртуальной таблицы, параметры которой можно задать с помощью конструкторов класса.

При размещении компонентов внутри ячеек таблицы все они получают одинаковые размеры. Если один из параметров, задающих размерность таблицы, равен нулю, это означает, что соответствующий столбец или строка может содержать любое количество элементов.

Конструкторы класса `GridLayout`:

– создание сетки с заданным количеством строк `rows` и столбцов `cols` — `public GridLayout(int rows, int cols);`

– создание таблицы с заданным количеством строк `rows` и столбцов `cols` и с заданным горизонтальным и вертикальным зазором `hgap` и `vgap` между компонентами — `public GridLayout(int rows, int cols, int hgap, int vgap).`

Компоненты размещаются менеджером `GridLayout` слева направо в строках таблицы в том порядке, в котором они заданы в методах `add()`. Ниже представлен исходный код приложения с пятью

кнопками, размещенными с помощью менеджера GridLayout. Результат работы программы показан на рисунке 1.6.

```
import java.awt.*;
import java.awt.event.*;
class GridTest  extends Frame {
  GridTest(String s) {
    super(s);
    setLayout(new GridLayout(2, 3, 5, 5));
    add(new Button("Кнопка 1"));
    add(new Button("Кнопка 2"));
    add(new Button("Кнопка 3"));
    add(new Button("Кнопка 4"));
    add(new Button("Кнопка 5"));
    setSize(300, 100);
    setVisible(true);
  }
  public static void main(String[] args) {
    Frame f = new GridTest("Менеджер GridLayout");
    f.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent ev)
      {  System.exit(0); }
    });
  }
}
```

Рисунок 1.6 — Размещение кнопок менеджером GridLayout

1.4.4 Менеджер CardLayout

Менеджер компоновки CardLayout показывает в контейнере только один, верхний компонент. Остальные компоненты лежат под первым как игральные карты в колоде. Их расположение определяется

порядком, в котором написаны методы add(). Следующий компонент можно показать методом next(Container c), предыдущий — методом previous(Container c), последний — методом last(Container c), а первый — методом first(Container c). Аргумент этих методов — ссылка на контейнер, в который помещены компоненты, обычно this. Нужный компонент с именем name можно показать с помощью метода show(Container parent, String name).

В классе два конструктора:

– CardLayout() — не отделяет компонент от границ контейнера;

– CardLayout(int hgap, int vgap) — задает горизонтальные hgap и вертикальные vgap поля.

Менеджер CardLayout позволяет организовать произвольный доступ к компонентам. Метод add() для менеджера CardLayout имеет следующий вид:

add(Component comp, Object constraints),

где аргумент constraints должен иметь тип string и содержать имя компонента.

В примере, приведенном ниже, менеджер компоновки c1 работает с панелью p, помещенной в «центр» контейнера Frame. Панель p передается как аргумент parent в методах next() и show(). На «север» контейнера Frame добавлена панель p2 с меткой и раскрывающимся списком ch. Рисунок 1.7 демонстрирует результат работы программы.

```
import java.awt.*;
import java.awt.event.*;
class CardTest extends Frame {
  CardTest(String s) {
    super(s);
    Panel p = new Panel();
    CardLayout cl = new CardLayout();
    p.setLayout(cl);
    p.add(new Button("Русская страница"), "page1");
```

```
    p.add(new Button("English page"), "page2");
    p.add(new Button("Deutsche Seite"), "page3");
    add(p);
    cl.next(p);
    cl.show(p, "pagel");
    Panel p2 = new Panel();
    p2.add(new Label("Выберите язык:"));
    Choice ch = new Choice();
    ch.add("Русский");
    ch.add("Английский");
    ch.add("Немецкий");
    p2.add(ch);
    add(p2, BorderLayout.NORTH);
    setSize(400, 300);
    setVisible(true);
  }
  public static void main(String[] args) {
    Frame f = new CardTest(" Менеджер CardLayout");
    f.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent ev) {
        System.exit(0);
      }
    });
  }
}
```

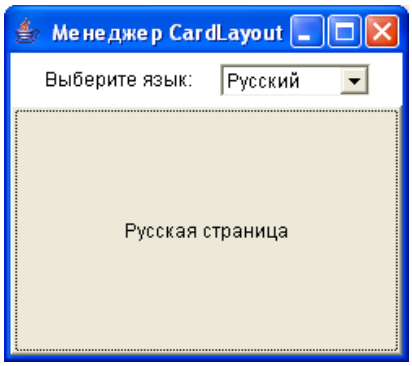


Рисунок 1.7 — Менеджер компоновки CardLayout

1.4.5 Менеджер GridBagLayout

Менеджер компоновки `GridBagLayout` позволяет размещать компоненты наиболее гибко, позволяя задавать различные размеры и положение для каждого компонента.

В классе `GridBagLayout` есть только один конструктор по умолчанию, без аргументов. Менеджер класса `GridBagLayout`, в отличие от других менеджеров компоновки, не содержит правил компоновки, так как он играет только организующую роль. Ему передаются ссылка на компонент и правила расположения этого компонента, а сам он помещает данный компонент по указанным правилам в контейнер. Все правила компоновки компонентов задаются в объекте класса `GridBagConstraints`.

Менеджер размещает компоненты в таблице с неопределенным заранее числом строк и столбцов. Один компонент может занимать несколько ячеек этой таблицы, заполнять ячейку целиком, располагаться в ее центре, углу или прижиматься к краю ячейки.

Класс `GridBagConstraints` содержит одиннадцать параметров, определяющих размеры компонентов, их положение в контейнере и взаимное положение, и несколько констант — значений некоторых полей. Они перечислены в таблице 1.1. Эти параметры определяются конструктором, имеющим одиннадцать аргументов. Второй конструктор — конструктор по умолчанию — присваивает параметрам значения, заданные по умолчанию.

Таблица 1.1 — Поля класса GridBagConstraints

Поле	Значение
anchor	Направление компоновки компонента в контейнере. Константы: CENTER, NORTH, EAST, NORTHEAST, SOUTHEAST, SOUTH, SOUTHWEST, WEST, и NORTHWEST; по умолчанию CENTER
fill	Растяжение компонента для заполнения ячейки. Константы: NONE, HORIZONTAL, VERTICAL, BOTH; по умолчанию NONE

Продолжение таблицы 1.1

Поле	Значение
gridheight	Количество ячеек в колонке, занимаемых компонентом. Целое типа int, по умолчанию 1. Константа REMAINDER означает, что компонент займет остаток колонки, RELATIVE – будет следующим по порядку в колонке
gridwidth	Количество ячеек в строке, занимаемых компонентом. Целое типа int, по умолчанию 1. Константа REMAINDER означает, что компонент займет остаток строки, RELATIVE – будет следующим в строке по порядку
gridx	Номер ячейки в строке. Самая левая ячейка имеет номер 0. По умолчанию константа RELATIVE, что означает: следующая по порядку
gridy	Номер ячейки в столбце. Самая верхняя ячейка имеет номер 0. По умолчанию константа RELATIVE, что означает: следующая по порядку
insets	Позволяет задать для компонента отступы от краев выделенной ему области. По умолчанию такие отступы отсутствуют.
ipadx, ipady	Горизонтальные и вертикальные поля вокруг компонентов; по умолчанию 0
weightx, weighty	Пропорциональное растяжение компонентов при изменении размера контейнера; по умолчанию 0, 0

Как правило, объект класса `GridBagConstraints` создается конструктором по умолчанию, затем значения нужных полей можно изменить, например:

```
GridBagConstraints gbc = new GridBagConstraints();
gbc.weightx = 1.0;
gbc.gridwidth = GridBagConstraints.REMAINDER;
gbc.gridheight =2;
```

После создания объекта `gbc` класса `GridBagConstraints` нужно указать менеджеру компоновки, что при помещении компонента `comp` в контейнер следует применять правила, занесенные в объект `gbc`. Для этого применяется метод `add(Component comp, GridBagConstraints gbc)`

Схема применения менеджера GridBagLayout такова:

```
// Создаем объект класса GridBagLayout
GridBagLayout gbl = new GridBagLayout();
// Добавляем его в контейнер
setLayout(gbl);
// Задаем правила компоновки по умолчанию
GridBagConstraints c = new GridBagConstraints();
// Создаем компонент
Button b1 = new Button();
// Меняем правила компоновки
c.gridwidth =2;
// Помещаем компонент b1 в контейнер
// по указанным правилам компоновки с
add(bl, c);
// Создаем следующий компонент
Button b2 = new Button();
// Меняем правила для его компоновки
c.gridwidth = 1;
// Помещаем в контейнер
add(b2, c);
```

Ниже приведен пример кода, размещающего на форме 5 кнопок с помощью менеджера компоновки GridBagLayout. Результат работы программы показан на рисунке 1.8.

```
import java.awt.*;
import java.awt.event.*;
class GridBagTest extends Frame{
  GridBagTest(String s) {
    super(s);
    setLayout(new GridBagLayout());
    GridBagConstraints        c        =        new
GridBagConstraints();
    GridBagConstraints        d        =        new
GridBagConstraints();
    c.gridwidth =2; // Меняем правила компоновки
    d.gridheight =4; //Меняем правила компоновки
```

```
    d.weightx = 10.0;
    // Помещаем компонент в контейнер
    add(new Button("Кнопка 1"), c);
    add(new Button("Кнопка 2"), d);
    add(new Button("Кнопка 3"));
    add(new Button("Кнопка 4"));
    add(new Button("Кнопка 5"));
setSize(300, 100);
    setVisible(true);
  }
  public static void main(String[] args) {
    Frame f=new GridBagTest("Менеджер GridBag");
    f.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent ev)
      {          System.exit(0);     }
    });
  }
}
```

Рисунок 1.8 — Пример компоновки компонентов менеджером GridBagLayout

1.4.6 Пример сложной компоновки

Для построения приложения, интерфейс которого приведен на рисунке 1.9, используются компоновки `FlowLayout`, `GridLayout` и `BorderLayout`. Для панели `Panel` используется менеджер компоновки по умолчанию — `FlowLayout`.

Программный код, реализующий эту задачу приведен ниже:

```
import java.awt.*;
import java.awt.event.*;
class BorderPanelTest extends Frame {
  BorderPanelTest(String s) {
```

```
    super(s);
// Создаем панель p2 с тремя кнопками,
//менеджер компоновки по умолчанию FlowLayout
    Panel p2 = new Panel();
    p2.add(new Button("Выполнить"));
    p2.add(new Button("Отменить"));
    p2.add(new Button("Выйти"));
// Создаем панель p3 с тремя кнопками
//менеджер компоновки GridLayout
    Panel p3 = new Panel();
    p3.setLayout(new GridLayout(3, 1));
    p3.add(new Button("Вариант 1"));
    p3.add(new Button("Вариант 2"));
    p3.add(new Button("Вариант 3"));
// Создаем панель p1
    Panel p1 = new Panel();
    p1.setLayout(new BorderLayout());
// Помещаем панель p2 с кнопками на "юге" панели p1
    p1.add(p2, BorderLayout.SOUTH);
// Помещаем панель p3 с кнопками на "востоке"
панели p1
    p1.add(p3, BorderLayout.EAST);
// Поле ввода помещаем на "севере"
    p1.add(new TextField("Поле ввода", 20),
BorderLayout.NORTH);
// Область ввода помещается в центре
    p1.add(new TextArea("Область ввода", 5, 20,
TextArea.SCROLLBARS_NONE),
           BorderLayout.CENTER);
    add(new Scrollbar(Scrollbar.HORIZONTAL),
BorderLayout.SOUTH);
    add(new Scrollbar(Scrollbar.VERTICAL),
BorderLayout.EAST);
// Панель p1 помещаем в "центре" контейнера
    add(p1, BorderLayout.CENTER);
```

```
//задаем размер окна
    setSize(400, 300);
    setVisible(true);
  }
public static void main(String[] args) {
    Frame f = new BorderPanelTest("Сложная
компоновка");
    f.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent ev) {
        System.exit(0);
      }
    }
    );
  }
}
```

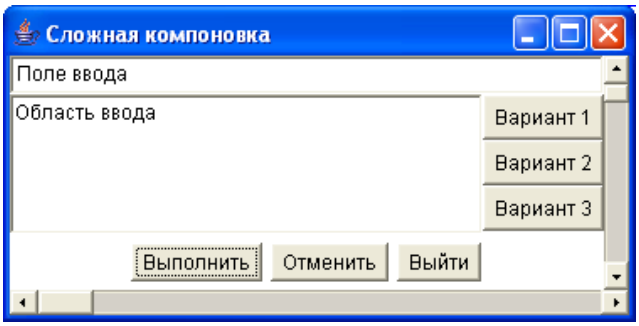


Рисунок 1.9 — Компоновка с помощью FlowLayout, GridLayout и BorderLayout

1.5 Задание к лабораторной работе

Наберите код, представленный ниже. Скомпилируйте и запустите программу. Проанализируйте ее работу и занесите в отчет результаты. Запишите в отчет, какие менеджеры компоновки используются в программе, а также их настройки.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class Test extends JFrame implements
ActionListener{
JButton main_btn;
```

```
String cur_font;
int cur_size=14;
Test(String s) {
    super(s);
    Container contentPane = this.getContentPane();
    contentPane.setLayout(new GridLayout(3, 3, 15,
15));
    JButton font_btn1 = new JButton("Arial");
    font_btn1.addActionListener(this) ;
    contentPane.add(font_btn1);
    JButton font_btn2 = new JButton("Times New
Roman");
    font_btn2.addActionListener(this) ;
    contentPane.add(font_btn2);
    JButton font_btn3 = new JButton("Courier New");
    font_btn3.addActionListener(this) ;
    contentPane.add(font_btn3);
    JButton size_up = new JButton("Увеличить
шрифт");
    size_up.addActionListener(this) ;
    contentPane.add(size_up);
    main_btn = new JButton("Hellow world!");
    main_btn.addActionListener(this) ;
    contentPane.add(main_btn);
    JButton size_down = new JButton("Уменьшить
шрифт");
    size_down.addActionListener(this) ;
    contentPane.add(size_down);
    JButton color_btn1 = new JButton("Красный");
    color_btn1.addActionListener(this) ;
    contentPane.add(color_btn1);
    JButton color_btn2 = new JButton("Зеленый");
    color_btn2.addActionListener(this) ;
    contentPane.add(color_btn2);
    JButton color_btn3 = new JButton("Желтый");
```

```
    color_btn3.addActionListener(this) ;
    contentPane.add(color_btn3);
    setSize(600, 250);
    setVisible(true);
    }
public void actionPerformed(ActionEvent e){
    Container contentPane = getContentPane();
    if       (e.getActionCommand().equals("Увеличить
шрифт"))
    {   cur_size+=2;
        main_btn.setFont(new
java.awt.Font(cur_font, 0, cur_size));
    }
    else    if
(e.getActionCommand().equals("Уменьшить шрифт"))   {
        cur_size-=2;
        main_btn.setFont(new
java.awt.Font(cur_font, 0, cur_size));
        }
    else  if  (e.getActionCommand().equals("Arial")){

        cur_font="Arial";
        main_btn.setFont(new  java.awt.Font("Arial",
0, cur_size));
    }
    else    if  (e.getActionCommand().equals("Times
New Roman")) {
    cur_font="Times New Roman";
        main_btn.setFont(new   java.awt.Font("Times
New Roman", 0, cur_size));
    }
    else    if
(e.getActionCommand().equals("Courier New"))   {
    cur_font="Courier New";
```

```
            main_btn.setFont(new  java.awt.Font("Courier
New", 0, cur_size));
        }
        else if
(e.getActionCommand().equals("Красный"))
            main_btn.setBackground(Color.red);
        else if
(e.getActionCommand().equals("Зеленый"))
            main_btn.setBackground(Color.green);
        else if (e.getActionCommand().equals("Желтый"))
                main_btn.setBackground(Color.yellow);
        else
                System.out.println("Ошибка!");
}
  public static void main(String[] args) {
    Frame f = new Test("Лабораторная работа №1");
    f.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent ev)
      {  System.exit(0); }
    });
  }
}
```

1.6 Контрольные вопросы

1. Понятие компонента.
2. Понятие контейнера.
3. Опишите основные подходы к размещению компонентов в Java.
4. Как создать простое приложение в Java?
5. Класс Component. Назначение. Методы.
6. Класс Container. Назначение. Методы.
7. Как добавить компонент в контейнер?
8. Как разместить компоненты в окне приложения вручную?
9. Контейнер Pane. Назначение. Конструктор.
10. Контейнер ScrollPane. Назначение. Конструктор. Методы.
11. Контейнер Window. Назначение. Конструктор. Методы. События.

12. Контейнер Frame. Назначение. Конструктор. Методы. События.

13. Контейнер Dialog. Назначение. Конструктор. Методы. События.

14. Контейнер FileDialog. Назначение. Конструктор. Методы. События.

15. Опишите работу менеджера компоновки FlowLayout.

16. Конструкторы менеджера компоновки FlowLayout.

17. Опишите работу менеджера компоновки BorderLayout.

18. Конструкторы менеджера компоновки BorderLayout.

19. Опишите работу менеджера компоновки GridLayout.

20. Конструкторы менеджера компоновки GridLayout.

21. Опишите работу менеджера компоновки GridBagLayout.

22. Конструкторы менеджера компоновки GridBagLayout.

23. Класс GridBagConstraints. Параметры.

24. Как с помощью GridBagConstraints разместить компонент в нескольких ячейках виртуальной таблицы?

24. Опишите работу менеджера компоновки CardLayout.

25. Конструктор менеджера компоновки CardLayout.

2 ЛАБОРАТОРНАЯ РАБОТА №2

Тема работы: Java. Библиотека визуальных компонентов Swing

Цель работы: научиться основам работы с библиотекой визуальных компонентов Swing.

2.1 Общие сведения о библиотеке Swing

Для реализации оконного интерфейса, снабженного такими элементами, как меню, кнопки, списки, в Java используется библиотека Swing.

Программы, взаимодействующие с пользователем, т.е. воспринимающие сигналы от клавиатуры и мыши, работают в графической среде. Каждое приложение, предназначенное для работы в графической среде, должно создать хотя бы одно окно, в котором будет происходить его работа, и зарегистрировать его в графической оболочке операционной системы, чтобы окно могло взаимодействовать с операционной системой и другими окнами: перекрываться, перемещаться, менять размеры, сворачиваться в ярлык.

Приложения Java должны работать в любой или хотя бы во многих графических средах. В каждой графической оболочке это делается по-своему, с помощью библиотек данной операционной системы. Такие интерфейсы были названы peer-интерфейсами.

Библиотека классов Java, основанных на peer-интерфейсах, получила название AWT (Abstract Window Toolkit). При выводе объекта, созданного в приложении Java и основанного на peer-интерфейсе, на экране создается парный ему (peer-to-peer) объект графической подсистемы операционной системы, который и отображается на экране. Эти объекты тесно взаимодействуют во время работы приложения. Поэтому графические объекты AWT в каждой графической среде имеют вид, характерный для этой среды.

Основное понятие графического интерфейса пользователя (GUI) — компонент (component) графической системы. Это может быть поле для ввода текста, кнопка, строка меню, полоса прокрутки, переключатель и т.п. Само окно приложения — тоже компонент. Компоненты могут быть видимыми и невидимыми, как, например, панель, объединяющая группу компонентов. Компонентом

считается объект класса Component или объект всякого класса, расширяющего класс Component. В классе Component собраны общие методы работы с любым компонентом графического интерфейса пользователя. Этот класс основной в библиотеке AWT.

Каждый компонент перед выводом на экран помещается в контейнер — объект класса Container или его подкласса. Прямой наследник этого класса — класс JComponent — вершина иерархии многих классов библиотеки Swing.

Основное окно приложения, активно взаимодействующее с операционной системой, необходимо построить по правилам графической системы. Оно должно перемещаться по экрану, изменять размеры, реагировать на действия мыши и клавиатуры. В окне должны быть, как минимум, следующие стандартные компоненты:

– строка заголовка (title bar), с левой стороны которой рекомендуется разместить кнопку контекстного меню, а с правой – кнопки сворачивания и разворачивания окна и кнопку закрытия приложения;

– необязательная строка меню (menu bar) с раскрывающимися («выпадающими») пунктами меню;

– горизонтальная и вертикальная полосы прокрутки (scrollbars);

– рамка (border) окна, реагирующая на действия мыши (при изменении размера).

2.2 Состав библиотеки Swing

Библиотеку Swing можно разделить на четыре раздела.

Раздел 1 — компоненты для ввода и отображения информации:

– текстовая метка JLabel;

– однострочное текстовое поле JTextField;

– многострочное текстовое поле JTextArea;

– списки JComboBox и JList;

– другие.

Раздел 2 — компоненты для управления работой программы:

– кнопка JButton;

– флажок JCheckBox;

– переключатель JRadioButton и группа переключателей ButtonGroup;

– главное меню JMainMenu;

– контекстное меню JPopupMenu;

– панель инструментов JToolBar;

– и другие.

Раздел 3 — компоненты для работы с базами данных.

Раздел 4 — прочие компоненты.

Общее количество компонентов (элементов управления) в библиотеке Swing на настоящий момент более 50, и она постоянно дополняется.

2.3 Свойства элементов управления Swing

Ниже представлены свойства, общие для большинства компонентов, называемых элементами управления (ЭУ).

Многие свойства являются закрытыми членами классов ЭУ. Поэтому доступ к ним можно получить с использованием set и get методов. Примеры установки свойств:

```
jLabel2.setText("Пароль");
jLabel4.setFont(new java.awt.Font("Tahoma", 0, 14));
rbTest.setBackground(Color.yellow);
```

Свойство name отображает имя ЭУ в программе. Имя — это уникальный идентификатор, используя который, можно обращаться к ЭУ программно (задавать значения свойств, вызывать методы, обрабатывать события).

Свойства horizontalTextPosition, verticalTextPosition задают способ выравнивания текста внутри ЭУ. Возможные значения: LEFT (по левому краю), CENTER (по центру), RIGHT (по правому краю), LEADING, TRAILING.

Свойство background задает фоновый цвет ЭУ.

Свойство border отображает внешний вид границы ЭУ.

Свойство editable (логическое) задает возможность редактирования текста, содержащегося в ЭУ.

Свойство enabled (логическое) управляет доступом к ЭУ.

Свойство font задает параметры шрифта для ЭУ: тип, размер, начертание. Цвет шрифта задает свойство foreground.

Свойство margin определяет отступ для текста внутри ЭУ: margin = {Left, Top, Width, Height}. Сам текст, который отображается в ЭУ, содержится в свойстве text.

Перечисленные выше свойства имеются у большинства ЭУ. Специфические свойства будут рассмотрены по мере изучения каждого ЭУ.

Свойства maximumSize и minimumSize определяют максимальный и минимальный размеры элемента управления, до которых его может увеличивать / уменьшать менеджер компоновки.

Обратите внимание, что при установке некоторых свойств используются константы (например, yellow) или методы (например, setFont) других классов.

2.4 Методы элементов управления Swing

Большинство методов ЭУ относятся к рассмотренным выше методам set и get, обеспечивающим доступ к свойствам. Недостаточно объявить ЭУ в классе проекта, его еще нужно добавить с помощью метода add.Метод add предназначен для добавления ЭУ в контейнер (например, класс приложения this или панель panel).

Синтаксис метода:

```
object.add(component, property),
```

где object — имя объекта-контейнера;

component — имя добавляемого компонента (ЭУ);

property — значение свойства добавляемого компонента.

2.5 События элементов управления Swing

2.5.1 События и приемники событий

Когда пользователь нажимает клавишу на клавиатуре или мышью на элементе управления, создается объект-событие, который пересылает другому объекту-приемнику (слушателю) сообщение.

В классе GUI следует указать (зарегистрировать), какой объект (или объекты) является слушателем. Объект-слушатель будет реагировать на события. Кроме того, в GUI должны быть определены

методы, которые будет вызывать программа при пересылке события слушателю.

Например, оператор `stopButton.addActionListener(this);` регистрирует объект `this` как слушатель (Listener) для получения событий от кнопки `stopButton`. Модификатор `this` указывает, что слушателем является сам класс, в котором описана кнопка `stopButton`.

2.5.2 Обработка событий

Для обработки событий следует определять соответствующие классы приемников. Так, например, кнопка генерирует события действия (`ActionEvent`), которые обрабатываются слушателями событий действия (`ActionListener`).

Объекты слушателей событий имеют тип `xxxListener`, где `xxx` — имя класса событий. Чтобы превратить некоторый класс в класс слушателя событий типа `xxxListener`, необходимо в заголовке определения класса добавить `implements xxxListener`, а в коде определить метод — обработчик события.

Ниже представлен пример кода, реализующего интерфейс слушателя событий `ActionListener` и метод для его обработки — `actionPerformed`:

```
1  import javax.swing.*;
2  import java.awt.*;
3  import java.awt.event.*;
4  public class ButtonDemo extends JFrame implements
ActionListener
5  {
6  public static final int WIDTH = 300;
7  public static final int HEIGHT = 200;
8  // Создание и отображение окна класса ButtonDemo
9  public static void main(String[] args)
10 {
11 ButtonDemo buttonGui = new ButtonDemo();
12 buttonGui.setVisible(true);
13 }
```

```
14 public ButtonDemo()
15 {
16 setSize(WIDTH, HEIGHT);
17 addWindowListener(new WindowDestroyer());
18 setTitle("Кнопки и события");
19 Container contentPane = getContentPane();
20 contentPane.setBackground(Color.blue);
21 contentPane.setLayout(new FlowLayout());
22 JButton stopButton = new JButton("Красный");
23 stopButton.addActionListener(this) ;
24 contentPane.add(stopButton);
25 JButton goButton = new JButton("Зеленый");
26 goButton.addActionListener(this);
27 contentPane.add(goButton);
28 }
29 public void actionPerformed(ActionEvent e)
30 {
31 Container contentPane = getContentPane();
32 if (e.getActionCommand().equals("Красный"))
33 contentPane.setBackground(Color.red);
34 else
35 if (e.getActionCommand().equals("Зеленый"))
36 contentPane.setBackground(Color.green);
37 else
38 System.out.println("Ошибка!");
39 }
40 }
```

В 1–3 строках с помощью ключевого слова `import` производится подключение пакетов javax.swing (компоненты Swing), java.awt (компоненты AWT), java.awt.event (обработчики событий AWT). Знак * в конце объявлений означает, что в программу подключаются все элементы (класс, константы, объявления и т.д.) пакета.

Четвертая строка — это объявление класса `ButtonDemo`, базирующегося на классе окна JFrame и способного обрабатывать события (`implements ActionListener`).

В 6 и 7 строках объявлены две константы: ширина и высота окна в пикселях соответственно.

С 9 по 13 строки расположено описание главной функции приложения – функции `main`. В 11 и 12 строках инициализируется основной класс программы – класс `ButtonDemo`, описание которого приведено в строках с 14 по 28-ю.

В 16-й строке устанавливается размер окна, а в 18-й задается его заголовок.

В 17-й строке подключается класс слушателя `WindowDestroyer`, предназначенный для корректного закрытия окна.

В 19-й строке в качестве контейнера определяется панель `contentPane`, в 20-й строке устанавливается ее цвет (синий), а в 21-й — тип компоновки (плавающая - `FlowLayout`).

В 22-й строке объявляется кнопка с именем `stopButton` и надписью «Красный», а в 23-й строке указывается слушатель для этой кнопки `this`. Использование this в качестве слушателя говорит о том, что последним является сам класс `ButtonDemo`. В 24-й строке кнопка заносится в контейнер.

С 25-й по 27-ю строки аналогичные действия выполняются для второй кнопки с именем `goButton` и надписью «Зеленый».

В строках 29-39 приведено описание обработчика события. Сообщение о событии передается в объект `e` типа `ActionEvent`.

Внутри обработчика проверяется, какой объект вызвал событие с помощью метода `getActionCommand`. Если событие вызвала кнопка с надписью «Красный», то устанавливается красный цвет фона контейнера, если же кнопка с надписью «Зеленый», то устанавливается зеленый цвет фона контейнера

Наберите текст программы, сохраните его на диск под именем **ButtonDemo.java**.

Для того, чтобы окно приложения правильно закрывалось, создадим собственный класс. В файл с именем **WindowDestroyer.java** запишите класс слушателя окна:

```
// Класс слушателя событий для окна
import java.awt.*;
```

```
import java.awt.event.*;
public class WindowDestroyer
extends WindowAdapter {
    public void windowClosing(WindowEvent e)
    { System.exit(0);}
}
```

Наберите в командной строке и выполните следующие команды:

```
javac.exe WindowDestroyer.java
javac.exe ButtonDemo.java
java.exe ButtonDemo
```

В результате на экране Вы увидите окно, приведенное на рисунке 2.1. Протестируйте работу этой программы.

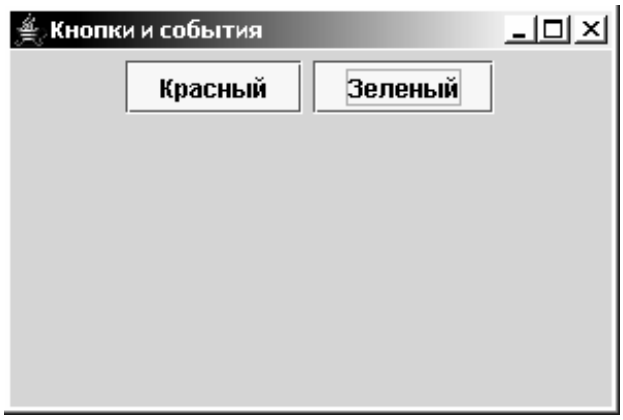


Рисунок 2.1 — Внешний вид окна программы ButtonDemo

2.5.3 Стандартные события и их обработка

Большинство событий, генерируемых ЭУ, относятся к классу `Event`. Большая часть событий связана с обработкой действий пользователя: манипуляций с мышью (события мыши) и с клавиатурой (события клавиатуры).

Двумя наиболее важными методами обработки событий мыши являются `mouseDown()` и `mouseUp()`. Обработчик `mouseDown()` вызывается при каждом нажатии кнопки мыши, а `mouseUp()` — когда кнопка мыши отпускается.

События мыши перечислены в таблице 2.1. Синтаксис всех методов следующий:

```
boolean mouseEventName(Event событие, int x, int y),
```

где mouseEventName – имя события;

событие – объект, описывающий данное событие;

x, y – координаты указателя мыши в момент генерации события.

После обработки события все методы должны возвращать true.

Таблица 2.1 — События мыши

Событие	Описание
mouseDown	Вызывается при нажатии кнопки мыши
mouseUp	Генерируется, когда кнопка мыши отпускается
mouseMove	Вызывается при перемещении мыши. Событие генерируется до тех пор, пока мышь перемещается в окне и ее кнопка не нажата
mouseDrag	Вызывается в случае перемещения мыши при нажатой кнопке. Генерируется до тех пор, пока мышь перемещается в окне и ее кнопка нажата
mouseEnter	Вызывается при помещении указателя мыши на окно
mouseExit	Вызывается при выводе указателя мыши из окна

Если окно владеет фокусом ввода, то при нажатии клавиши генерируется событие клавиатуры keyDown(), а при ее отпускании — соответственно keyUp():

```
boolean keyDown(Event объект, int клавиша),
boolean keyUp(Event объект, int клавиша),
```

где объект описывает событие, а клавиша содержит код нажатой клавиши. Параметр клавиша может иметь тип char. При перекрытии этих методов приложение должно возвращать true в случае обработки события.

2.5.4 Слушатели событий и класс ActionEvent

Методы обработки событий описаны в интерфейсах-слушателях. У каждого события, кроме inputEvent, есть свой интерфейс. Имена интерфейсов составлены из имени события и слова Listener, например, ActionListener, MouseListener. Классы, реализующие такой интерфейс, называются слушателями (listeners): они «слушают», что происходит в объекте, чтобы отследить возникновение события и обработать его.

Чтобы связаться с обработчиком события, классы-источники события должны получить ссылку на экземпляр `eventHandler` класса-обработчика события одним из методов `addXxxListener(XxxEvent eventHandler)`, где `Xxx` – имя события. Такой способ регистрации, при котором слушатель оставляет сведения источнику для своего вызова при наступлении события, называется обратный вызов (callback). Обратное действие — отказ от обработчика, прекращение прослушивания — выполняется методом `removeXxxListener()`.

Таким образом, компонент-источник, в котором произошло событие, сам не занимается его обработкой. Он обращается к экземпляру класса-слушателя, умеющего обрабатывать события, делегирует ему полномочия по обработке. Такая схема удобна тем, что можно сменить класс-обработчик и обработать событие по-другому или назначить несколько обработчиков одного и того же события. С другой стороны, можно на прослушивание нескольких объектов-источников событий назначить один обработчик.

В метод-обработчик события передается параметр типа `событие`. Чаще всего используется класс `ActionEvent`. Так, например, для кнопки `jButton1` этот метод выглядит так:

```
void jButton1_actionPerformed(ActionEvent e)
```

Класс `ActionEvent` содержит информацию о состоянии мыши и клавиатуры на момент возникновения события, а также дополнительную информацию об источнике события.

В таблице 2.2 перечислены методы класса `ActionEvent`. Для определения состояния клавиш используется метод `KeyEvent.getKeyModifiersText`.

Таблица 2.2 — Методы класса ActionEvent

Метод	Описание
e.getActionCommand()	Строка, связанная с командой
e.getModifiers()	Состояние клавиш
e.getID()	Получить код команды
e.getWhen()	Определить время запуска команды
e.getSource()	Имя элемента, вызвавшего событие

2.5.5 Генерация и посылка событий

Классы Java могут не только принимать события, но и возбуждать (генерировать) их. Например:

```
void jButton1_actionPerformed(ActionEvent e)
{
String msg = «Это сообщение для другого класса»;
ActionEvent event=new ActionEvent(this,1,msg);
otherActionPerformed(event);
}
```

Приведенный фрагмент кода представляет обработчик события нажатия ЛКМ на кнопке `jButton1`. Вначале создается строка `msg`, которая будет отправлена клиенту. Затем создается объект-экземпляр класса `ActionEvent`. Именно он будет передан клиенту при вызове метода `otherActionPerformed`:

```
protected void otherActionPerformed(ActionEvent e)
{
. . .
((ActionListener)listeners.elementAt(i)).actionPerf
ormed(e);
. . .
}
```

У класса `ActionEvent` три конструктора. В нашем примере был использован следующий:

```
public ActionEvent(Object source, int id, String
command)
```

Первый параметр — экземпляр объекта, который посылает сообщение. Второй параметр — код, который Вы присваиваете событию. Использование кода позволяет в дальнейшем определять источник и назначение посланного сообщения. Третий параметр — строка, содержащая само сообщение.

2.5.6 События, связанные с окнами

При проведении операций с окнами возникает событие `windowEvent`. Класс содержит следующие методы (табл. 2.3).

Таблица 2.3 — Методы класса windowEvent

Метод	Что обрабатывает
public void windowOpened(WindowEvent e)	Открытие
public void windowClosing(WindowEvent e)	Закрытие
public void windowClosed(WindowEvent e)	Закрыто
public void windowIconified(WindowEvent e)	Свертывание
public void windowDeiconified(WindowEvent e)	Развертывание
public void windowActivated(WindowEvent e)	Получение фокуса
public void windowDeactivated(WindowEvent e)	Потеря фокуса

Аргумент `e` этих методов содержит ссылку на окно-источник типа `window`, которую можно получить, используя метод `e.getwindow()`.

2.5.7 События, связанные с компонентами

При проведении операций с компонентами возникает событие `ComponentEvent`. Этот класс содержит следующие методы (табл. 2.4).

Таблица 2.4 — Методы класса ComponentEvent

Метод	Что обрабатывает
public void componentResized(ComponentEvent e)	Изменен размер
public void componentMoved(ComponentEvent e)	Перемещение
public void componentShown(ComponentEvent e)	Появление
public void componentHidden(ComponentEvent e)	Исчезновение

Аргумент `e` этих методов содержит ссылку на компонент, вызвавший событие, которую можно получить, используя метод `e.getComponent()`.

2.5.8 События, связанные с фокусом

При передаче фокуса возникает событие `FocusEvent`. Этот класс содержит следующие методы (табл. 2.5).

Таблица 2.5 — Методы класса FocusEvent

Метод	Что обрабатывает
public void focusGained(FocusEvent e)	Фокус получен
public void focusLost(FocusEvent e)	Фокус утрачен

Аргумент `e` этих методов содержит ссылку на компонент, вызвавший событие, которую можно получить, используя метод `e.getComponent()`.

2.5.9 Создание нескольких слушателей для одного компонента

Как указывалось ранее, один класс — «слушатель» может «прослушивать» сразу несколько компонентов. Однако гораздо чаще встречается обратная ситуация — несколько слушателей должны следить за одним компонентом.

К каждому компоненту можно присоединить сколько угодно слушателей одного и того же события или разных типов событий. Однако при этом не гарантируется какой-либо определенный порядок их вызова, хотя чаще всего слушатели вызываются в порядке написания методов `addXxxListener()`.

Если нужно задать определенный порядок вызовов слушателей для обработки события, то придется обращаться к ним друг из друга или создавать объект, вызывающий слушателей в нужном порядке.

Ссылки на присоединенные методами `addXxxListener()` слушатели можно было бы хранить в любом классе-коллекции, например, `vector`, но в пакет java.awt специально для этого введен класс `AWTEventMuiticaster`. Он реализует все одиннадцать интерфейсов `XxxListener`, значит, сам является слушателем любого события. Основу класса составляют статические методы `add`, написанные для каждого типа событий, например: `add(ActionListener a, ActionListener b)`.

Эти методы возвращают ссылку на тот же интерфейс, в данном случае, `ActionListener`, и присоединяют объект `a` к объекту b, создавая совокупность слушателей одного и того же типа. Это позволяет использовать их наподобие операций `a += b`.

Для событий типа inputEvent, а именно, KeyEvent и MouseEvent, есть возможность прекратить дальнейшую обработку события методом consume(). Если записать вызов этого метода в класс-слушатель, то следующие слушатели не будут обрабатывать событие. Этим способом обычно пользуются, чтобы отменить стандартные действия компонента, например, «вдавливание» кнопки.

2.6 Компоненты для ввода и отображения информации, входящие в библиотеку Swing

2.6.1 Текстовая метка JLabel

Текстовая метка предназначена для отображения информации в виде строки текста. Используя свойства, рассмотренные в п. 2.3, можно управлять внешним видом метки (рис. 2.2).

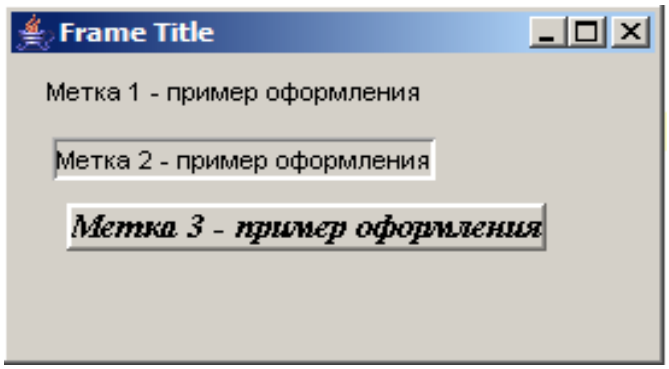


Рисунок 2.2 — Примеры оформления меток

Текст, который отображается в метке, задан в свойстве text. Получить его значение можно методом getText().

У объекта JLabel определено несколько конструкторов:

– JLabel() — пустой объект без текста;

– JLabel(String text) — объект с текстом text, который прижимается клевому краю компонента;

– JLabel(String text, int alignment) — объект с текстом text и выравниванием alignment текста, задаваемого одной из трех констант: CENTER, LEFT и RIGHT. Метод getAlignment() позволяет изменить выравнивание текста.

В классе JLabel происходят события класса Component: ComponentEvent, FocusEvent, KeyEvent, MouseEvent.

2.6.2 Текстовые поля JTextField и JTextArea

Текстовые поля `JTextField` (однострочное текстовое поле) и `JTextArea` (многострочное текстовое поле) представляют собой простейшие текстовые редакторы (рисунок 2.3).

JTextField JTextArea

Рисунок 2.3 — Текстовые поля

Их назначение — ввод и отображение текста. Как и все текстовые редакторы, эти ЭУ для манипуляций с текстом поддерживают комбинации горячих клавиш: **Ctrl+C**, **Ctrl+V**, **Ctrl+X**, **Ctrl+Z** и др. В таблице 2.6 перечислены общие свойства для этих ЭУ.

Таблица 2.6 — Общие свойства JTextField и JTextArea

Свойство	**Описание**	**Допустимые значения**
caretColor	Цвет текстового курсора	
caretPosition	Текущая позиция текстового курсора	
disabledTextColor	Цвет заблокированного текста	
editable	Разрешить редактирование	True/False
selectedTextColor	Цвет выделенного текста	
text	Содержимое текстового поля	

У текстового поля `JTextArea` есть ряд специфических свойств, перечисленных в таблице 2.7. Основной конструктор класса `JTextArea`:

```
JTextArea(String text, int rows, int columns).
```

Он создает область ввода с текстом `text`, числом видимых строк `rows` и числом колонок `columns`.

Конструктор по умолчанию `JTextArea()` создает пустое поле шириной в одну колонку.

Таблица 2.7 — Свойства компонента JTextArea

Свойство	**Описание**	**Допустимые значения**
columns, rows	Количество столбцов и строк	
lineWrap	Разрешить перенос слов в строке	true/false
tabSize	Количество символов, на которые переходит курсор при нажатии клавиши табуляции	

В классе `JTextArea` определены следующие методы:

– `append(string text)`, добавляющий текст `text` в конец уже введенного текста;

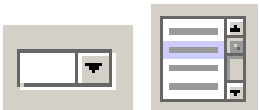
– `insert(string text, int pos)`, вставляющий текст в указанную позицию `pos`;

– `replaceRange(String text, int begin, int end)`, удаляющий текст, начиная с индекса `begin` по `end` включительно, и помещающий вместо него текст `text`.

Кроме событий класса `Component`, таких, как: `ComponentEvent`, `FocusEvent`, `KeyEvent`, `MouseEvent`, в текстовых полях при изменении текста пользователем происходит событие `TextEvent`, а при нажатии на клавишу Enter в поле `TextField` — событие `ActionEvent`.

2.6.3 Списки JComboBox и JList

В библиотеке Swing представлены два вида списка: `JComboBox` и `JList` (рисунок 2.4).



JComboBox JList

Рисунок 2.4 — Списки

Список `JComboBox` — это раскрывающийся список, а `JList` — раскрытый список. Позиции в списках нумеруются, начиная с нуля. В таблице 2.8 перечислены общие свойства списков.

Таблица 2.8 — Общие свойства JComboBox и JList

Свойство	**Описание**
border	Вид окантовки
selectedIndex	Номер выбранного элемента списка
selectedItem	Выбранный элемент списка

У класса `JList` определено четыре конструктора:

– `JList()` — создает пустой список;

– `JList(Vector listData)` — создает список с элементами из вектора `listData`;

– `JList(Object[] listData)` — создает список с элементами из массива `listData`.

– `JList(ListModel dataModel)` — создает список, использующий интерфейс `dataModel`.

Особенности моделей списков `JList` описаны в интерфейсе `ListModel`. Данный интерфейс требует реализации четырех методов. Два метода служат для присоединения и удаления слушателей событий, происходящих при обновлении данных списка; один метод возвращает элемент, находящийся на некоторой позиции списка; еще один метод позволяет списку узнать, сколько данных в данный момент содержит модель.

Во многих ситуациях достаточно стандартной модели, поставляемой вместе с библиотекой Swing, она называется `DefaultListModel`. Фактически модель эта представляет собой динамический массив, который способен оповещать об изменениях в себе заинтересованных слушателей.

Для добавления и удаления элементов списка необходимо вызывать методы объекта `ListModel`:

– `add(int index, Object element)` — вставляет объект `element` на позицию с номером `index`;

– `addElement(Object element)` — вставляет объект `element` в конец списка;

– `remove(int index)` или `removeElementAt(int index)` — удаляет элемент, стоящий на позиции `index`;

– removeAllElements() или clear() — удаляет все элементы из списка.

Для получения количества элементов в списке необходимо вызвать метод getSize().

Ниже приведен пример формы со списком, в который при нажатии на кнопку «Обновить» динамически добавляются элементы:

```
import javax.swing.*;
import java.awt.event.*;
public class UsingListModel extends JFrame {
//модель
    private DefaultListModel dlm;
    public UsingListModel() {
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        // заполнение модели данными
        dlm = new DefaultListModel();
        dlm.add(0, "Первый");
        dlm.add(0, "Второй");
        dlm.add(0, "Третий");
        // создаем кнопку и список
        JPanel contents = new JPanel();
        JButton add = new JButton("Обновить");
        add.addActionListener(new  ActionListener()
    {
            public void actionPerformed(ActionEvent
        e) {
                dlm.add(0, "Новый!");
                validate();
            }
        });
        JList list1 = new JList(dlm);
        // добавляем компоненты
        contents.add(add);
        contents.add(new JScrollPane(list1));
        setContentPane(contents);
        setSize(400, 200);
```

```
        setVisible(true);
    }
    public static void main(String[] args) {
        new UsingListModel();
    }
}
```

Вызов для окна метода `validate()` изменяет интерфейс в соответствие с новыми размерами списка.

Список `JList` включает используемую по умолчанию модель выделения `DefaultListSelectionModel`. Данная модель поддерживает три режима выделения элементов списка и предоставляет полную информацию о текущем выделении. В таблице 2.9 представлены режимы выделения элементов списка.

Таблица 2.9 — Режимы выделения элементов JList

Режим	Описание
SINGLE_SELECTION	Выделение одного элемента списка.
SINGLE_INTERVAL_SELECTION	Выделение нескольких смежных элементов списка.
MULTIPLE_INTERVAL_SELECTION	Выделение нескольких элементов списка в произвольном порядке.

Программисту доступны следующие методы модели выделения элементов списка:

– `setSelectedIndex(int idx)` — выделение элемента списка;

– `setSelectionInterval(int anchor, int lead)` — выделение нескольких смежных элементов;

– `addSelectionInterval(int anchor, int lead)` — добавление к уже имеющемуся выделению еще одного интервала выделения;

– `setSelectedIndices(int[] rows)` — выделение нескольких элементов списка в произвольном порядке.

Методы `setSelectionInterval()` и `addSelectionInterval()` в качестве параметров используют позиции первого и последнего выделяемых элементов. Если они совпадают, то выделяется один элемент. Необязательно, чтобы первое число было больше второго. В модели выделения хранятся начало интервала выделения `anchor` и конец интервала выделения `lead`. Один элемент списка может быть выбран щелчком мыши или нажатием клавиш управления курсором. Интервалы и дополнительные элементы могут быть выбраны с помощью вспомогательных клавиш `Shift` и `Ctrl`.

В таблице 2.10 представлены методы для получения информации о выделенных элементах.

Таблица 2.10 — Методы получения информации из списка

Метод	Назначение
isSelectedlndex()	Функция проверки наличия выделенных элементов списка.
getSelectedlndex(), getSelectedIndices()	Функции получения позиции первого выделенного элемента (если выделений нет, то возвращается -1) или массив позиций выделенных элементов.
getSelectedValue(), getSelectedValues()	Получение выбранного элемент списка или массива элементов.

Конструкторы `JComboBox()` идентичны конструкторам элемента `JList`.

Метод `addElement(Object el)` добавляет в список новые элементы. Они располагаются в порядке написания методов `addElement()` и нумеруются с нуля.

Вставить новый элемент в позицию списка с номером `position` можно методом `insertElementAt(Object el, int position)`.

Удалить один элемент из списка можно методом `removeElement(Object el)` или `removeElementAt(int`

position), а все элементы сразу — методом removeAllElements().

Метод getSize() возвращает количество элементов в списке.

Выяснить, какой именно элемент находится в позиции pos, можно методом getElementAt(int pos).

Определение индекса выбранного пункта производится методом getSelectedIndex(), а его содержимого — методом getSelectedItem().

2.7 Компоненты для управления работой программы

2.7.1 Кнопка JButton

Кнопка JButton служит для инициации какого-либо действия или серии действий (рис. 2.5).

Рисунок 2.5 — Кнопка JButton

В классе JButton определены два конструктора, первый из которых позволяет создавать кнопку без надписи, а второй — кнопку с надписью:

```
public JButton();
public JButton(String label);
```

Основные методы кнопки перечислены в таблице 2.11

Таблица 2.11 — Методы кнопки JButton

Метод	Действие
public String getLabel();	Получение надписи на кнопке
protected String paramString();	Получение строки параметров, отражающей состояние кнопки
public void setLabel(String label);	Установка надписи на кнопке

2.7.2 Флажок JCheckBox

Флажок `JCheckBox` (рис. 2.5) предназначен для задания параметров программы и реализации множественного выбора пользователя.

Рисунок 2.5 — Компонент JCheckBox

Три конструктора `JCheckBox()`, `JCheckBox(String label)`, `JCheckBox(String label, boolean state)` создают компонент без надписи, с надписью `label` в состоянии `off` (не установлен), и в заданном состоянии `state`. Методы доступа `getLabel()`, `setLabel(String label)`, `getState()`, `setState (boolean state)` возвращают и изменяют указанные параметры флажка.

Компоненты `JCheckBox` удобны для быстрого и наглядного выбора из списка, целиком расположенного на экране.

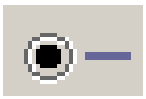
Методы компонента `JCheckBox` перечислены в таблице 2.12.

Таблица 2.12 — Методы компонента JCheckBox

Метод	Действие
public CheckboxGroup getCheckboxGroup();	Получение группы, к которой относится данный переключатель с зависимой фиксацией
public String getLabel();	Получение названия переключателя
public boolean getState();	Определение текущего состояния переключателя
protected String paramString();	Получение строки параметров
public void setCheckboxGroup (CheckboxGroup g);	Установка группы, к которой относится данный переключатель с зависимой фиксацией
public void setLabel(String label);	Установка названия переключателя
public void setState(boolean state);	Установка нового состояния переключателя

2.7.3 Переключатель JRadioButton и группа ButtonGroup

Переключатель `JRadioButton` (рис. 2.6) предназначена для фиксации выбора пользователя. Так же, как и `JCheckBox`, она может находиться во включенном или выключенном состоянии.


— JRadioButton

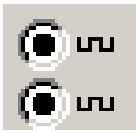

— ButtonGroup

Рисунок 2.6 — Переключатель JRadioButton и группа кнопок ButtonGroup

Одиночный переключатель создается с помощью одного из конструкторов:

– `public jRadioButton()` — переключатель без надписи;

– `public jRadioButton(String text)` — переключатель с надписью `text`;

– `public jRadioButton(String text, Boolean state)` — переключатель с надписью `text`, находящийся в состоянии `state`.

Основные методы переключателя перечислены в таблице 2.13.

Таблица 2.13 — Методы компонента JRadioButton

Метод	Действие
public String getText();	Получение надписи на переключателе
protected String paramString();	Получение строки параметров
public void setText(String label);	Установка надписи на переключателе
public void setSelected(boolean state);	Установка нового состояния переключателя

Группа переключателей создается с помощью конструктора `public ButtonGroup()`.

2.8 Проектирование интерфейса

2.8.1 Выбор менеджера компоновки

Первым этапом проектирования интерфейса программы является выбор менеджера компоновки. Возможные виды компоновок рассмотрены нами в лабораторной работе № 1.

2.8.2 Контейнеры

Контейнеры предназначены для компоновки элементов управления в окне программы. Базовым классом для всех контейнеров служит класс `Container`, все остальные являются его потомками (табл. 2.14). В таблице 2.15 перечислены основные методы контейнеров.

Таблица 2.14 — Контейнеры AWT и Swing. Подклассы класса Container

Класс	Описание
Panel, JPanel	Реализация простейшего контейнера
Applet, JApplet	Служит для создания апплетов. Подкласс класса Panel
Window, JWindow	Простой контейнер без меню и обрамления. Используется редко
Dialog, JDialog	Подкласс класса Window. Диалоговые окна (модальные и немодальные)
FileDialog, JFileChooser	Подкласс класса Dialog. Диалоговые окна для работы с файлами
Frame, JFrame	Подкласс класса Window. Используется в приложениях. Может содержать меню

Таблица 2.15 — Основные методы контейнеров

Метод	Описание
add(Component)	Добавить компонент в контейнер
remove(Component)	Удалить компонент из контейнера
setLayout(LayoutManager)	Установить тип компоновки
getLayout()	Определить тип компоновки

Использование нескольких контейнеров в приложении позволяет расширить возможности компоновки (компоновки) элементов управления в окне.

2.8.3 Размещение элементов управления

При размещении элементов управления на форме важным является выбор контейнеров и менеджеров компоновки.

На рисунке 2.7 приведен пример приложения, в котором используются различные менеджеры компоновки, а в Приложении А — программный код примера.

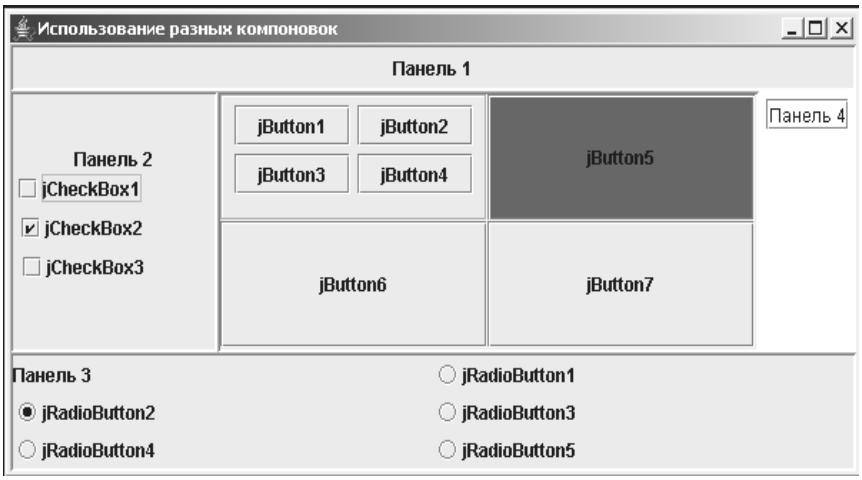


Рисунок 2.7 — Использование различных менеджеров компоновки в одном приложении

Создав объект класса `Component` или его наследника, его добавляют к предварительно созданному объекту класса `Container` или его подкласса одним из методов `add()`. Класс `Container` является невидимым компонентом, он расширяет класс `Component`. Таким образом, в контейнер наряду с компонентами можно помещать другие контейнеры, в которых находятся другие компоненты, достигая тем самым большей гибкости расположения компонентов.

2.8.4 Программирование слушателей событий

Одну и ту же команду можно выполнить разными способами. Пользователь может выбрать пункт меню, нажать соответствующую клавишу или нажать по кнопке панели инструментов. В этом случае очень удобна модель событий, при которой все эти события связывают с одним и тем же обработчиком. Пусть `blueAction` — это объект некоторого класса (например, `ColorAction`), реализующего интерфейс `ActionListener` и изменяющего цвет фона окна на синий. Можно связать этот объект с разными источниками событий:

– кнопкой с надписью **Blue** на панели инструментов;

– пунктом меню **Blue**;

– нажатием клавиш **<Ctrl+B>**.

Команда, изменяющая цвет, должна выполняться одинаково, независимо от того, что именно привело к ее выполнению — нажатие на кнопку или клавишу, выбор пункта меню.

В пакете **Swing** предусмотрен механизм, позволяющий инкапсулировать команды и связывать их с несколькими источниками событий — интерфейс `Action`. Действие (`action`) — это объект, инкапсулирующий описание команды (в виде текстовой строки или пиктограммы) и параметры, необходимые для выполнения программы (например, нужный цвет).

Интерфейс `Action` содержит следующие методы:

- `void actionPerformed(ActionEvent event);`
- `void setEnabled(boolean b);`
- `boolean isEnabled();`
- `void putValue(String key, Object value);`
- `Object getValue(String key);`
- `void addPropertyChangeListener (PropertyChangeListener listener);`
- `void removePropertyChangeListener (PropertyChangeListener listener).`

Первый метод похож на метод интерфейса `ActionListener`, так как интерфейс `Action` расширяет интерфейс `ActionListener`. Следовательно, вместо объекта класса, реализующего интерфейс `ActionListener`, можно использовать объект класса, реализующего интерфейс `Action`.

Следующие два метода позволяют блокировать и разблокировать действие, а также проверить, является ли указанное действие разблокированным (т.е. доступным) в данный момент. Если пункт меню или кнопка панели инструментов связаны с заблокированным действием, они выделяются светло-серым цветом.

Методы `putValue()` и `getValue()` позволяют записывать и извлекать из памяти произвольные пары имя-значение объектов класса, реализующего интерфейс `Action`.

Предопределенные строки, такие как `Action.NAME` и `Action.SMALL_ICON`, содержат имена и пиктограммы объектов действий, например:

```
Action.putValue(Action.NAME, «Blue»);
Action.putValue(Action.SMALL_ICON,
new ImageIcon(«blue-ball.gif»));
```

В таблице 2.16 приведены предопределенные имена.

Таблица 2.16 — Предопределенные имена действий

Имя	Значение
NAME	Имя действия. Отображается на кнопке и в названии пункта меню
SMALL_ICON	Место для хранения пиктограммы, которая изображается на кнопке, в пункте меню или на панели инструментов
SHORT_DESCRIPTION	Краткое описание пиктограммы, отображается в строке подсказки
LONG_DESCRIPTION	Подробное описание пиктограммы. Может использоваться для подсказки. Не используется ни одним компонентом из библиотеки Swing
MNEMONIC_KEY	Мнемоническое сокращение. Отображается в пункте меню
ACCELERATOR_KEY	Используется для хранения сочетания клавиш. Не используется ни одним компонентом из библиотеки Swing
ACTION_COMMAND_KEY	Ранее использовалось в теперь уже устаревшем методе registeredKeyboardAction()
DEFAULT	Может быть полезным для хранения разнообразных объектов. Не используется ни одним компонентом из библиотеки Swing

Если к меню или к панели инструментов добавляется какое-то действие, его имя и пиктограмма автоматически извлекаются из памяти и отображаются в меню и на панели. Значение `SHORT_DESCRIPTION` выводится в строке подсказки.

Последние два метода интерфейса Action позволяют извещать другие объекты, в частности меню и панели инструментов, об изменении свойств действий. Например, если меню создано как обработчик, предназначенный для отслеживания изменения свойств действия, и это действие впоследствии было заблокировано, то при отображении меню на экране соответствующее имя действия может быть выделено серым цветом.

Обратите внимание на то, что Action является интерфейсом, а не классом. Любой класс, реализующий этот интерфейс, должен обеспечивать семь методов, которые мы только что рассмотрели. Все они, кроме первого метода, содержатся в классе AbstractAction. Класс AbstractAction предназначен для хранения пар, состоящих из имен и значений, а также для управления обработчиками изменения свойств — для его практического использования необходимо создать подкласс и добавить метод actionPerformed().

Например, для создания действия, изменяющего цвет фона нужно поместить в память имя команды, соответствующую пиктограмму и желаемый цвет. Код цвета записать в таблицу, состоящую из пар имя-значение, предусмотренных классом AbstractAction.

Ниже приведен класс ColorAction. Конструктор задает пары, состоящие из имен и значений, а метод ActionPerformed() изменяет цвет фона.

```
public class ColorAction extends AbstractAction
{
public ColorAction(String name, Icon icon Color c){
    putValue(Action.NAME, name);
    putValue(Action.SMALL_ICON, icon);
    putValue("color", c);
    putValue(Action.SHORT_DESCRIPTION,  "Set  panel
color to " + name.toLowerCase());
}
public void actionPerformed(ActionEvent event) {
    Color c = (Color) getValue("color");
    setBackground(c);
} }
```

В программе необходимо создать объект класса `Action`:

```
Action blueAction = new ColorAction("Blue",
new ImageIcon("blue-ball.gif"), Color.BLUE);
```

Для того, чтобы связать действие с кнопкой, следует написать:

```
JButton blueButton = new JButton(blueAction);
```

Этот конструктор считывает имя и пиктограмму действия, помещает короткое описание в строку подсказки и регистрирует объект `Action` в качестве обработчика.

Допустим, нужно связать действия с нажатием клавиш. Сообщения о нажатии клавиш передаются компонентам, обладающим фокусом ввода. Пусть в программе есть панель с тремя кнопками. В любой момент одна из этих трех кнопок может владеть фокусом. Каждая из этих кнопок должна обрабатывать события, связанные с нажатием клавиш, и реагировать, например, на сочетания клавиш **<Ctrl+Y>**, **<Ctrl+B>** и **<Ctrl+R>**.

В версии 1.2 пакета JDK были предложены два решения, позволяющих связать клавиши с действиями: метод `registerKeyboardAction()` класса `JComponent` и механизм отображения клавиш для команд `JTextComponent`. В версии JDK 1.3 решения были объединены. Для того чтобы связать действия с нажатием клавиш, сначала нужно создать объект класса `KeyStroke` — это класс, инкапсулирующий описание клавиш.

Чтобы создать объект этого класса, не нужно вызывать конструктор. Вместо него следует использовать статический метод `getKeyStroke()` класса `KeyStroke`. После этого необходимо указать виртуальный код клавиш и флаги (например, для сочетаний, в состав которых входят клавиши `Ctrl` или `Alt`):

```
KeyStroke ctrlBKey = KeyStroke.getKeyStroke
      (KeyEvent.VK_B, InputEvent.CTRL_MASK);
```

Существует также способ, позволяющий описывать нажатие клавиши в виде строки:

```
KeyStroke ctrlBKey =
KeyStroke.getKeyStroke("ctrl B");
```

Каждый объект класса JComponent содержит три условия ввода, связывающих объекты класса KeyStroke с действиями. Эти отображения соответствуют различным условиям фокуса элемента.

При нажатии клавиши отображения ввода обрабатываются в следующем порядке:

1. Проверяется константа WHEN_FOCUSED компонента, владеющего фокусом ввода. Если предусмотрена реакция на нажатие клавиши, выполняется соответствующее действие, если действие не заблокировано, процесс прекращается.

2. Начиная с компонента, содержащего фокус ввода, выполняется проверка условия WHEN_ANCESTOR_OF_FOCUSED_COMPONENT его родительского компонента. Как только условие будет выполнено, производится соответствующее действие. Если действие не заблокировано, процесс прекращается.

3. Проверяются все видимые и незаблокированные компоненты в окне, обладающем фокусом ввода. Компоненты должны быть зарегистрированы с константой WHEN_IN_FOCUSED_WINDOW. Каждый из этих компонентов (в порядке регистрации) получает возможность выполнить соответствующее действие. Когда будет выполнено первое незаблокированное действие, процесс прекратится. Условие ввода извлекается из компонента с помощью метода getInputMap():

```
InputMap imap = panel.getInputMap
(JComponent.WHEN_FOCUSED);
```

Константа WHEN_FOCUSED означает, что условие проверяется, если компонент обладает фокусом ввода. Это условие не проверяется, если фокусом владеет одна кнопка, а не панель в целом. Каждая из оставшихся двух констант также позволяет изменить цвет фона в ответ на нажатие клавиш.

Класс InputMap не связывает напрямую объекты класса KeyStroke с объектами класса ColorAction, реализующего интерфейс Action. Вместо этого объектам класса KeyStroke он ставит в соответствие произвольные объекты, а с помощью второго отображения, реализованного классом ActionMap, связывает их с

действиями. Это облегчает связывание одного и того же действия с разными клавишами, зарегистрированными при разных условиях ввода.

Итак, каждый компонент имеет три условия ввода и одно условие, связывающее нажатие клавиш с действиями. Чтобы объединить их, нужно присвоить каждому действию имя.

```
imap.put(KeyStroke.getKeyStroke("ctrl Y"),
"panel.yellow");
ActionMap amap = panel.getActionMap();
amap.put("panel.yellow", yellowAction);
```

Если нужно задать «пустое» действие, можно использовать константу `none`. Это позволяет легко заблокировать клавишу.

```
imap.put(KeyStroke.getKeyStroke("ctrl C", "none");
```

В документации JDK рекомендуется, чтобы название клавиш и соответствующего действия совпадали. Такое решение вряд ли можно назвать оптимальным. Имя действия отображается на кнопке и в пункте меню. Оно может изменяться в процессе разработки, в частности, это неизбежно при адаптации пользовательского интерфейса для разных языков. Поэтому лучше задавать имена действий, выводящихся на экран, не зависящими от их настоящих имен.

Итак, чтобы одна и та же операция выполнялась в ответ на нажатие кнопки, выбор пункта меню или нажатие клавиши, следует сделать следующее:

– создать класс, расширяющий класс `AbstractAction`. Один класс можно будет использовать для программирования разных действий;

– создать объект класса, реализующего интерфейс `Action`;

– создать кнопку или пункт меню на основе объекта класса, реализующего интерфейс `Action`;

– для действий, которые выполняются в ответ на нажатие клавиш, нужно сделать один дополнительный шаг. Сначала следует идентифицировать компонент окна верхнего уровня, например панель, содержащую все остальные компоненты;

– проверить константу на значение `WHEN_ANCESTOR_OF_FOCUSED_COMPONENT` для компонента верхнего уровня. Создать объект класса `KeyStroke` для нужного

нажатия клавиш. Создать объект, соответствующий нажатию клавиш, например строку, описывающую нужное действие. Добавить пару (нажатие клавиши + действие) к отображению ввода;

– получить информацию о соответствии действий и клавиш для компонента верхнего уровня. Добавить пару (клавиша + действие) в отображение.

Ниже приведена программа, задающая соответствие между кнопками, нажатием клавиш и действиями. Действия выполняются в ответ на активизацию кнопок или нажатие клавиш **<Ctrl+Y>**, **<Ctrl+B>** или **<Ctrl+R>**.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class ActionTest {
public static void main(String[] args) {
    ActionFrame frame = new ActionFrame();
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_C
LOSE);
    frame.setVisible(true);
} }
/** Фрейм с панелью, демонстрирующий изменение
цвета. */
class ActionFrame extends JFrame {
public ActionFrame() {
    setTitle("ActionTest");
    setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
    // Добавление панели к фрейму.
    ActionPanel panel = new ActionPanel();
    add(panel);
}
public static final int DEFAULT_WIDTH = 300;
public static final int DEFAULT_HEIGHT = 200;
}
/** Панель, цвет которой можно изменять щелчком на
кнопке или нажатием клавиш. */
```

```
class ActionPanel extends JPanel {
public ActionPanel() {
    // Определение действий.
    Action yellowAction = new ColorAction("Yellow",
new ImageIcon("yellow-ball.gif"), Color.YELLOW);
    Action blueAction = new ColorAction("Blue",
new ImageIcon("blue-ball.gif"), Color.BLUE);
    Action redAction = new ColorAction("Red",
new ImageIcon("red-ball.gif"),Color.RED);
    // Кнопки для выполнения операций.
    add(new JButton(yellowAction));
    add(new JButton(blueAction));
    add(new JButton(redAction));
    // Связывание клавиш Y, B и R с именами.
    InputMap imap =
getInpMap(JComponent.WHEN_ANCESTOR_OF_FOCUSED_COMPO
NENT);
    imap.put(KeyStroke.getKeyStroke("ctrl      Y"),
"panel.yellow");
    imap.put(KeyStroke.getKeyStroke("ctrl      B"),
"panel.blue");
    imap.put(KeyStroke.getKeyStroke("ctrl      R"),
"panel.red");
    // Связывание имен с действиями
    ActionMap amap = getActionMap();
    amap.put("panel.yellow", yellowAction);
    amap.put("panel.blue", blueAction);
    amap.put("panel.red", redAction);
}
public class ColorAction extends AbstractAction {
    /** Создание действий для изменения цвета.
    @param name Надпись на кнопке
    @param icon Пиктограмма на кнопке
    @param c Цвет фона */
```

```
    public ColorAction(String name, Icon icon,
Color c)    {
    putValue(Action.NAME, name);
    putValue(Action.SMALL_ICON, icon);
    putValue(Action.SHORT_DESCRIPTION,
"Set panel color to " + name.toLowerCase());
    putValue("color", c);
}
public void actionPerformed(ActionEvent event)
{
    Color c = (Color) getValue("color");
    setBackground(c);
}
```

2.9 Печать из java-программ

2.9.1 Основные принципы печати

Печать на принтере возможна для любого наследника класса `java.awt.Component`. Для этого в классе `Component` предусмотрен специальный метод `public void print(Graphics g)` предназначенный для печати внешнего вида компонента. Что именно будет выводиться на печать, программист должен описать самостоятельно, переопределив этот метод. Переопределение метода `print()` очень похоже на переопределение метода `public void paint(Graphics g)`, использующегося для рисования внешнего вида компонента на экране.

2.9.2 Печать на принтере

Рисование в графическом контексте принтера ничем не отличается от рисования в графическом контексте экрана, и при этом используются те же методы и классы, однако в процессе подготовки изображения для принтера важно учитывать размеры и расположение области печати. В противном случае часть изображения, не уместившаяся в области печати, будет обрезана.

Для определения размеров листа необходимо использовать следующие методы класса `PrintJob`:

– `public Dimension getPageDimension()` — определяет размеры листа в пикселях, что удобно при рисовании в графическом контексте принтера;

– `public int getPageResolution()` — определяет «разрешение» листа в пикселях на дюйм. Отметим, что такое разрешение не имеет отношения к физическому разрешению принтера.

Большинство принтеров при печати оставляют поля на листе, образующиеся в силу особенностей механики конкретного принтера, не позволяющих подвести печатающую головку к самому краю листа (рис. 2.8).

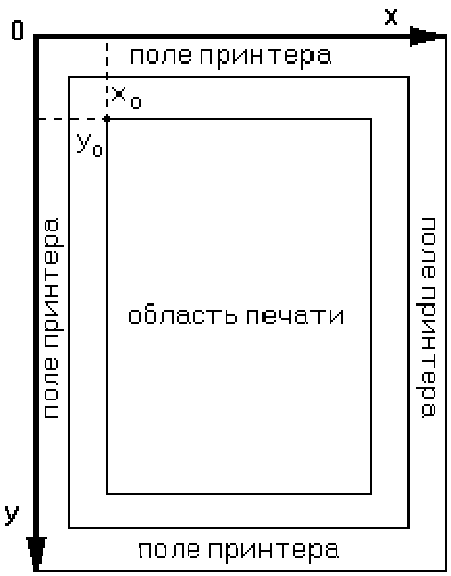


Рисунок 2.8 — Параметры листа бумаги

Поля принтера могут отличаться друг от друга по размеру. В пакете `java.awt.print.*` находятся классы, описывающие исключительные ситуации, возникающие при печати.

Для формирования изображения необходимо создать класс, реализующий интерфейс `java.awt.print.Printable`. От этого интерфейса создаваемый класс наследует единственный абстрактный метод `public int print(Graphics graphics, PageFormat pageFormat, int pageIndex) throws PrinterException`.

Переопределяя этот метод, программист задает, что именно будет печататься принтером. Метод `print` получает от вызывающей его программы ссылку на графический контекст принтера. Объект типа

Graphics, передаваемый методу print() в качестве аргумента, также реализует интерфейс java.awt.print.PrinterGraphics и метод, описанный в этом интерфейсе public PrinterJob getPrinterJob(), который позволяет внутри метода print() получить ссылку на объект типа PrinterJob.

Помимо ссылки на графический контекст принтера, метод print() также получает ссылку на объект типа java.awt.print.PageFormat, позволяющий получить информацию о размерах области печати (с учетом или без учета полей принтера) и ориентации бумаги, выбранной в настройках принтера.

Третий аргумент, получаемый методом print() — номер страницы для печати. Отсчет страниц ведется от 0. В случае если запрашиваемая страница может быть сформирована методом print() он должен вернуть константу Printable.PAGE_EXISTS, в противном случае — Printable.NO_SUCH_PAGE.

Ниже приводится пример класса, формирующего изображения страниц для печати. Этот класс позволит распечатать 2 страницы, отобразив на них их номера:

```
class PageImage implements Printable{
    public int print(Graphics g, PageFormat pf, int
pageNumber)
      throws PrinterException{
        if(pageNumber>1){
        return(Printable.NO_SUCH_PAGE);}
            Else
{g.drawString("Page "+(pageNumber+1),150,150);}
        return(Printable.PAGE_EXISTS);
    }
}
```

При формировании изображения большое значение имеют его размеры, если изображение не попадает в область печати, то принтер не выполнит задание. Причем поля, изначально установленные для документа отличаются от размеров полей системного принтера (рис. 2.8).

Для определения области печати и ориентации листа необходимо воспользоваться методами объекта типа `PageFormat` (табл. 2.17).

Таблица 2.17 — Методы класса PageFormat

Метод	Действие
public double getHeight()	Возвращает полную высоту листа в пикселях из расчета 72 пикселя на дюйм
public double getWidth()	Возвращает полную ширину листа в пикселях
public double getImageableHeight()	Возвращает высоту области печати в пикселях
public double getImageableWidth()	Возвращает ширину области печати в пикселях
public double getImageableX()	Смещение по горизонтали левого верхнего угла области печати относительно левого верхнего угла листа
public double getImageableY()	Смещение по вертикали левого верхнего угла области печати относительно левого верхнего угла листа
public int getOrientation()	Возвращает ориентацию листа в виде целочисленной константы: PageFormat.PORTRAIT, PageFormat.LANDSCAPE, PageFormat.REVERSE_LANDSCAPE

Также в классе `PageFormat` определены методы для смены текущих настроек принтера при помощи вспомогательного класса `java.awt.print.Paper`.

После описания класса, формирующего изображение для печати, необходимо связать его с принтером и отпечатать создаваемый в нем документ. Для связывания принтера с объектом типа `Printable`, формирующим изображение для печати, в классе `PrinterJob` существует метод `public void setPrintable(Printable painter)`, а для печати документа используется метод `public void print() throws PrinterException` того же класса. Таким образом для печати документа можно использовать следующий код:

```
printerJob.setPrintable(new PageImage());
try {
    printerJob.print();
```

```
    Thread.sleep(1000); // Задержка на 1 секунду
}
catch(Exception e){
    System.err.println(e.getMessage());
}
```

Временная задержка в 1 секунду использована на случай, если операционная система, в которой выполняется программа, не сразу поставит задание в очередь на печать. При работе этого кода никаких окон появляться не будет, и он распечатает на принтере две страницы, сформированные в классе `PageImage`. Часто при написании подобных программ класс, осуществляющий печать документа, реализует в себе интерфейс `Printable` и переопределяет унаследованный метод `print()`, при этом программа получается более компактной.

2.9.3 Современные методы печати

Дополнительные классы и методы для печати, появившиеся в Java 1.4 API, существенно упрощают печать. Все дополнительные к Java 1.2 API классы для печати сгруппированы в пакете `javax.print` и его подпакетах.

Принципиальное отличие нового API заключается в том, что при печати учитывается тип содержимого документа, задаваемый при помощи стандартных, определенных в RFC-документах 2045 и 2046, MIME-типов (Multipurpose Internet Mail Extensions Types). Существует достаточно много MIME-типов форматирования данных, которые принимает Java 1.4 API, например: «text/plain» — текстовые ASCII документы, «text/html» — документы формата HTML, «application/pdf» — документы формата PDF, «application/postscript» — документы формата PostScript, «image/gif» и «image/jpeg» — соответствующие графические форматы, и др.

Для обозначения данных такого типа в англоязычной литературе используют термин «**service formatted print data**», а для данных стандартных MIME-типов, извлекаемых из некоторого источника, где они уже находились в предварительно отформатированном виде используют термин «**client formatted print data**».

Практически для осуществления печати документа, с помощью Java 1.4 API, необходимо выполнить следующую последовательность действий:

– получить список доступных системе принтеров;

– создать объект для дальнейшего взаимодействия с принтером;

– описать тип форматирования (MIME-тип) печатаемых данных;

– задать свойства печати;

– установить связь с источником данных;

– распечатать данные.

2.9.4 Получение списка принтеров

Для идентификации системного принтера используют объект, реализующий интерфейс `javax.print.PrintService`. Получение ссылки на этот объект производится с помощью методов класса `PrintServiceLookup`: `public static PrintService lookupDefaultPrintService()` — осуществляет поиск принтера по умолчанию, в случае если такой принтер отсутствует, возвращает `null`; `lookupPrintServices(DocFlavor flavor, AttributeSet attributes)` — ищет все доступные операционной системе принтеры, удовлетворяющие заданным требованиям (например, способных печатать документы формата A3). Если в качестве аргументов указать значения `null`, то вернется массив из всех принтеров, доступных в операционной системе.

Пример кода, выполняющего описанное действие:

```
PrintService printService =
PrintServiceLookup.lookupDefaultPrintService();
```

2.9.5 Создание объекта для взаимодействия с принтером

Для взаимодействия с принтером используется объект типа `javax.print.DocPrintJob` по применению схожий с объектом `java.awt.print.PrinterJob`, применявшемся в Java 1.2 API. Ссылку на этот объект получают с помощью метода `public DocPrintJob createPrintJob()` объекта реализующего интерфейс `PrintService`.

Пример кода, выполняющего описанное действие:

```
DocPrintJob job=printService.createPrintJob();
```

2.9.6 Печать средствами Java2D

Расширенная графическая система Java 2D предлагает новые интерфейсы и классы для печати, собранные в пакет `java.awt.print`. Эти классы полностью перекрывают все стандартные возможности печати библиотеки AWT. Более того, они удобнее в работе и предлагают дополнительные возможности. Если этот пакет установлен в операционной системе, то нужно применять его, а не стандартные средства печати AWT.

Как и стандартные средства AWT, методы классов Java 2D выводят на печать содержимое графического контекста, заполненного методами класса `Graphics` или класса `Graphics2D`.

Всякий класс Java 2D, печатающий хотя бы одну страницу текста, графики или изображения называется классом, рисующим страницы (page painter). Такой класс должен реализовать интерфейс `Printable`, в котором описаны две константы и только один метод `print()`. Класс, «рисующий» страницы, должен реализовать этот метод. Метод `print()` возвращает целое значение типа `int` и имеет три аргумента:

```
print(Graphics g, PageFormat pf, int ind);
```

Первый аргумент `g` — это графический контекст, выводимый на лист бумаги, второй аргумент `pf` — экземпляр класса `PageFormat`, определяющий размер и ориентацию страницы, третий аргумент `ind` — порядковый номер страницы, начинающийся с нуля.

Метод `print()` класса, рисующего страницы, заменяет собой метод `paint()`, использовавшийся стандартными средствами печати AWT. Класс, рисующий страницы, не обязан расширять класс `Frame` и переопределять метод `paint()`. Все заполнение графического контекста методами класса `Graphics` или `Graphics2D` теперь выполняется в методе `print()`.

Когда печать страницы будет закончена, метод `print()` должен возвратить целое значение, заданное константой `PAGE_EXISTS`. Будет сделано повторное обращение к методу `print()` для печати следующей страницы. Аргумент `ind` при этом возрастет на 1. Когда

ind превысит количество страниц, метод print() должен возвратить значение NO_SUCH_PAGE, что служит сигналом окончания печати.

Система печати может несколько раз обратиться к методу paint() для печати одной и той же страницы. При этом аргумент ind не меняется, а метод print() должен создать тот же графический контекст.

Класс PageFormat определяет параметры страницы. На странице вводится система координат с единицей длины 1 пункт = 1/72 дюйма, начало которой и направление осей определяется одной из трех констант:

– PORTRAIT — начало координат расположено в левом верхнем углу страницы, ось x направлена вправо, ось y — вниз;

– LANDSCAPE — начало координат в левом нижнем углу, ось x идет вверх, ось y — вправо;

– REVERSE_LANDSCAPE — начало координат в правом верхнем углу, ось x идет вниз, ось y — влево.

Большинство принтеров не может печатать без полей, поэтому осуществляет вывод только в некоторой области печати (imageable area), координаты левого верхнего угла которой возвращаются методами getImageableX() и getImageableY(), а ширина и высота — методами getImageableWidth() и getImageableHeight().

Эти значения надо учитывать при расположении элементов в графическом контексте, например, при размещении строк текста методом drawString().

У класса только один конструктор по умолчанию — PageFormat(), задающий стандартные параметры страницы, определенные для принтера операционной системы по умолчанию.

Задать параметры страницы можно с помощью стандартного диалогового окна операционной системы. Метод pageDialog(PageDialog pd) открывает на экране стандартное окно «Параметры страницы» (Page Setup) операционной системы, в котором уже заданы параметры, определенные в объекте pd. Если пользователь выбрал в этом окне кнопку **<Отмена>**, то возвращается

ссылка на объект pd, если кнопку **<ОК>**, то создается и возвращается ссылка на новый объект. Объект pd в любом случае не меняется, обычно он создается конструктором.

Можно задать параметры страницы и из программы, но тогда следует сначала определить объект класса Paper конструктором по умолчанию:

```
Paper p = new Paper();
```

Затем методами p.setSize(double width, double height) и p.setImageableArea(double x, double y, double width, double height) нужно задать размер страницы и области печати. Затем следует определить объект класса pageFormat с параметрами по умолчанию:

```
PageFormat pf = new PageFormat();
```

И наконец задать новые параметры методом:

```
pf.setPaper(p);
```

Теперь вызывать на экран окно «Параметры страницы» методом pageDialog() уже не обязательно — произойдет фоновый процесс печати.

Далее надо создать задание на печать (print job) — указать количество страниц, их номера, порядок печати страниц, количество копий. Все эти сведения собираются в классе PrinterJob.

Система печати Java 2D различает два вида заданий. В более простых заданиях (Printable Job) есть только один класс, рисующий страницы, поэтому у всех страниц одни и те же параметры, страницы печатаются последовательно с первой по последнюю или с последней страницы по первую, это зависит от системы печати.

Второй, более сложный вид заданий (Pageable Job) определяет для печати каждой страницы свой класс, рисующий страницы, поэтому у каждой страницы могут быть собственные параметры. Кроме того, можно печатать не все, а только выбранные страницы, выводить их в обратном порядке, печатать на обеих сторонах листа. Для осуществления этих возможностей определяется экземпляр класса Book или создается класс, реализующий интерфейс Pageable.

В классе Book всего один конструктор, создающий пустой объект:

```
Book b = new Book();
```

После создания в данный объект добавляются классы, рисующие страницы. Для этого в классе Book есть два метода:

– append (Printable p, PageFormat pf) — добавляет объект p в конец;

– append(Printable p, PageFormat pf, int numPages) – добавляет numPages экземпляров p в конец; если число страниц заранее неизвестно, то задается константа UNKNOWN_NUMBER_OF_PAGES.

При составлении задания на печать, т. е. после создания экземпляра класса PrinterJob, надо указать вид задания одним из трех методов этого класса setPrintable(Printable pr), setPrintable (Printable pr, PageFormat pf) или setPageble (Pageable pg). Заодно задаются один или несколько классов pr, рисующих страницы в этом задании.

Остальные параметры задания можно задать в стандартном диалоговом окне **Печать** (Print) операционной системы, открывающемся при выполнении метода printDialog(). Указанный метод не имеет аргументов. Он возвратит true, когда пользователь щелкнет по кнопке <ОК>, и false после нажатия кнопки <Отмена>.

Если число копий больше 1, то используется метод setCopies(int n).

Метод defaultPage() класса PrinterJob возвращает объект класса PageFormat по умолчанию. Этот метод можно использовать вместо конструктора класса PageFormat. Создается экземпляр класса PrinterJob не конструктором, а статическим методом getPrinterJob().

Для начала печати вызывается метод print() класса PrinterJob. Этот метод не имеет аргументов. Он последовательно вызывает методы print(g, pf, ind) классов, рисующих страницы, для каждой страницы.

Ниже представлен исходный код программы, которая средствами Java2D печатает окружность и номер страницы.

```
import java.awt.*;
import java.awt.geom.*;
import java.awt.print.*;
class Print2Test implements Printable {
  public int print(Graphics g, PageFormat pf, int
ind) throws PrinterException {
// Печатаем не более 5 страниц
    if (ind > 4) {
      return Printable.NO_SUCH_PAGE;
    }
    Graphics2D g2 = (Graphics2D) g;
 g2.setFont(new Font("Serif", Font.ITALIC, 30));
 g2.setColor(Color.black);
 g2.drawString("Page " + (ind + 1), 100, 100);
 g2.draw(new Ellipse2D.Double(100, 100, 200, 200));
    return Printable.PAGE_EXISTS;
  } // конец класса Print2Test
  public static void main(String[] args) {
// 1.Создаем экземпляр задания
    PrinterJob pj = PrinterJob.getPrinterJob();
// 2.Открываем диалог «Параметры страницы»
    PageFormat pf = j.pageDialog(pj.defaultPage());
// 3. Задаем вид задания, объект класса,
//рисующего страницу, и выбр-е параметры стр.
    pj.setPrintable(new Print2Test(), pf);
// 4. Если нужно напечатать несколько копий, то:
    pj.setCopies(2);//по умолч. печат-ся 1 копия
// 5.Откр. диал. окно Печать (необязательно)
    if (pj.printDialog()) { // Если OK...
      try {
        pj.print();
      } // Обращается к print(g, pf, ind)
      catch (Exception e) {
```

```
            System.err.println(e);
        }
    }
// 6. Завершаем задание
    System.exit(0);
  }
}
```

2.10 Задание к лабораторной работе

1. Разработайте приложение для начисления заработной платы в отделе. Внешний вид формы показан на рисунке 2.9.

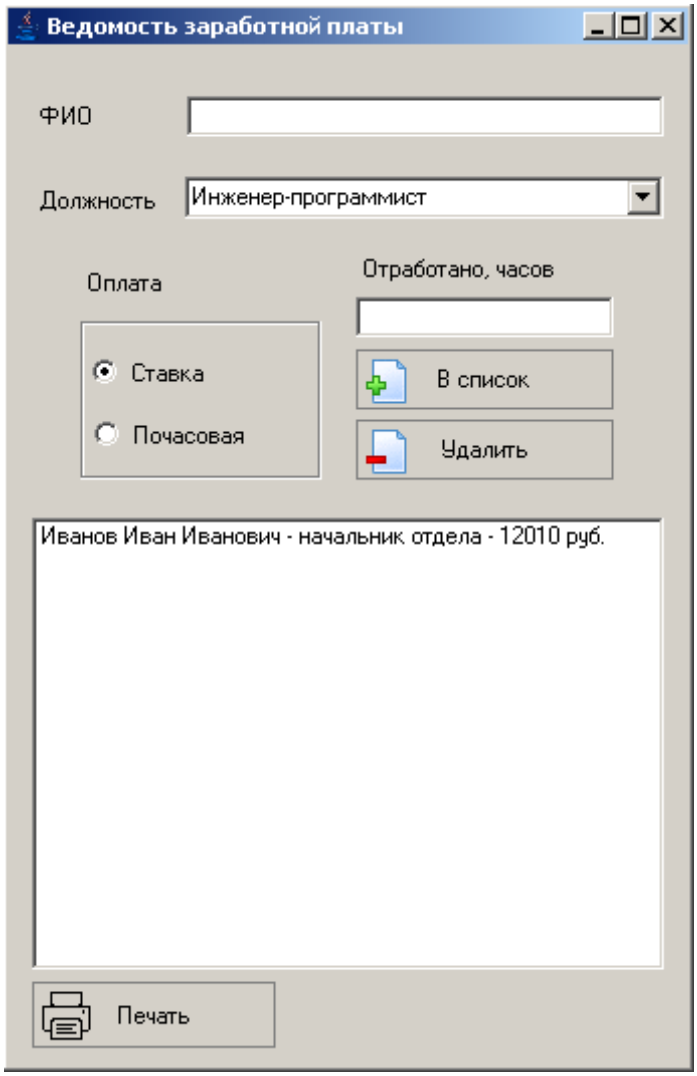


Рисунок 2.9 — Пример внешнего вида формы

2. При старте программа должна заполнять:

– список «Должности» набором из 5 должностей: инженер-программист, ведущий инженер, начальник отдела, инженер-системотехник, техник;

– массивы с данными о величинах окладов и почасовых ставок для соответствующих должностей.

3. Пользователь должен иметь возможность:

– выбрать должность из списка;

– выбрать вид оплаты: почасовая или ставка. Если выбрана почасовая оплата, поле «Отработано, часов» должно стать доступным для ввода;

– в поле «Количество часов» вводить числа с клавиатуры.

Если пользователь выбрал почасовую оплату, то сумму начисленной заработной платы рассчитывайте, как произведение отработанных часов и часовой ставки. Если пользователь выбрал оклад, эту величину следует принять в качестве начисленной суммы заработной платы.

После ввода ФИО, выбора должности и ввода данных о способе оплаты пользователь нажимает кнопку «В список». При этом данные о сотруднике и начисленной зарплате должны быть перенесены в список ниже в виде строки:

ФИО Должность Зарплата

При нажатии кнопки «Удалить» выделенные в списке строки должны быть удалены из списка.

При нажатии кнопки «Напечатать» список сотрудников должен выводиться на печать с заголовком **«Ведомость заработной платы»**. Внизу следует вывести итоговую сумму начисленной заработной платы.

4. Введите данные о 10 сотрудниках отдела с разными окладами и условиями работы. Выведите полученный список на печать.

5. Распечатайте текст программы и содержимое списка.

2.11 Контрольные вопросы

1. Для чего предназначены библиотеки AWT и Swing?
2. Какие компоненты входят в библиотеку Swing?
3. Перечислите общие свойства компонентов Swing.
4. Перечислите основные методы компонентов Swing.

5. Что такое событие? Что такое слушатель события?

6. Каким образом обрабатываются события?

7. Перечислите события мыши.

8. Перечислите события клавиатуры.

9. Перечислите события окна.

10. Перечислите события компонента.

11. Перечислите события фокуса.

12. Как зарегистрировать слушатель события?

13. Как назначить один слушатель событий на несколько компонентов?

14. Как назначить для одного компонента несколько слушателей событий?

15. Как определить, какой именно компонент вызвал событие?

16. Класс ActionEvent. Назначение, методы.

17. Как сгенерировать событие?

18. Текстовая метка JLabel. Свойства, методы, события.

19. Текстовое поле JTextField. Свойства, методы, события.

20. Текстовое поле JTextArea. Свойства, методы, события.

21. Список JList. Свойства, методы, события.

22. Раскрывающийся список JComboBox. Свойства, методы, события.

23. Кнопка JButton. Свойства, методы, события.

24. Флажок JCheckBox. Свойства, методы, события.

25. Переключатель JRadioButton. Свойства, методы, события.

26. Как организовать группу переключателей?

27. Менеджеры компоновки. Перечислите, назовите основные особенности.

28. Классы контейнеров. Для чего нужно несколько контейнеров в приложении?

29. Как задать параметры страницы?

30. Как вывести текст на принтер средствами Java?

31. Как вывести картинку на принтер средствами Java 2D?

3 ЛАБОРАТОРНАЯ РАБОТА №3

Тема работы: проектирование интерфейса программ в Java

Цель работы: научиться разрабатывать интерфейс пользовательских программ с использованием библиотек Swing и AWT.

3.1 Общие принципы разработки интерфейса

3.1.1 Требования к интерфейсу программ

Существует четыре основных критерия качества любого интерфейса:

– скорость работы пользователей;

– количество ошибок, которые совершают пользователи;

– время обучения пользователей;

– субъективное удовлетворение пользователей.

Скорость выполнения работы является важным критерием эффективности интерфейса.

Время выполнения работы пользователем состоит из интервалов: восприятия исходной информации, интеллектуальной работы (пользователь думает, что он должен сделать), физических действий пользователя и реакции системы. Как правило, длительность реакции системы является наименее значимым фактором.

Взаимодействие пользователя с программой состоит из семи шагов:

– формирование цели действий;

– определение общей направленности действий;

– определение конкретных действий;

– выполнение действий;

– восприятие нового состояния системы;

– интерпретация состояния системы;

– оценка результата.

Из этого списка видно, что процесс размышления занимает почти все время, в течение которого пользователь работает с программой, во всяком случае, шесть из семи этапов полностью заняты умственной деятельностью.

Длительность физических действий пользователя прежде всего зависит от степени автоматизации работы и степени необходимой

точности работы. Любое физическое действие, совершаемое человеком, может быть или точным, или быстрым. Вместе точность и быстрота встречаются исключительно редко, поскольку для этого нужно выработать существенную степень автоматизма.

Пользователь может управлять компьютером двумя способами, а именно мышью и клавиатурой. Клавиатура не требует особой точности движений — неважно, как быстро Вы нажали клавишу. Мышь, напротив, инерционное устройство — есть разница между медленным её перемещением и быстрым, сильным приложенным усилием и слабым. В свою очередь, мышь не предназначена для очень точных, в 1 или 2 пикселя, манипуляций, например, в графических программах всегда есть возможность перемещать объекты клавишами со стрелками. Именно поэтому любой маленький интерфейсный элемент будет всегда вызывать проблемы у пользователей.

Новые диалоговые окна стоит открывать не в центре экрана, а в центре текущего действия пользователя (если они не перекрывают важную информацию на экране).

Лучший способ повысить доступность кнопки — это сделать её большой и располагать ближе к курсору. Однако, чтобы ускорить нажатие кнопки, её, во-первых, можно сделать большего размера и, во-вторых, дистанцию к ней можно сделать нулевой.

Часто пользователи надолго прерывают свою работу. Помимо потери фокуса внимания, это плохо тем, что лишенная руководства система начинает простаивать. Это делает всегда верным следующее правило: если процесс предположительно будет длительным, система должна убедиться, что она получила всю информацию от пользователя до начала этого процесса.

Эту проблему можно решить так: система может считать, что если пользователь не ответил на вопрос, например, в течение пяти минут, то его ответ положительный. Таким образом, тот же самый сценарий решается по другому: пользователь отправляет документ на печать и уходит, система спрашивает «Вы уверены?» и ждет пять минут, после истечения этого времени она начинает печать.

3.1.2 Последовательность разработки интерфейса

В процессе разработки (дизайна) интерфейса можно выделить три основных этапа, а именно:

– первоначальное проектирование;

– создание прототипа;

– тестирование/модификация прототипа.

Процесс разработки всегда циклический: после тестирование возможен возврат к проектированию и т.д.

Рекомендуется выполнять разработку интерфейса сначала на бумаге, затем определиться с классами компонентов, которые необходимо использовать, а потом уже писать программу.

Основным этапом является проектирование (т.е. собственно дизайн), и проверка уже созданного интерфейса. Этап проектирования состоит из нескольких шагов, причем количество этих шагов довольно велико.

3.2 Классы для организации главного меню: JMenuBar, JMenu и JMenuItem

Элементы управления `JMenuBar`, `JMenu` и `JMenuItem` позволяют организовать в программе главное меню. Класс `JMenuBar` — это панель с меню, `JMenu` — пункты главного меню, а `JMenuItem` представляет подпункты (команды).

3.2.1 Добавление главного меню в программу

Для добавления меню в программу нужно создать экземпляры классов `JMenuBar`, `JMenu` и `JMenuItem`. Рассмотрим этот процесс на примере:

```
1  import javax.swing.*;
2  public class MenuDemo extends JFrame {
3       public static final int WIDTH = 600;
4       public static final int HEIGHT = 300;
5       public static void main(String[] args) {
6            MenuDemo md = new MenuDemo();
7            md.setVisible(true);  }
8  public MenuDemo()     {
9       addWindowListener(new WindowDestroyer());
```

```
10      setSize(WIDTH,HEIGHT);
11      setTitle("Пример главного меню");
12      JMenu myMenu = new JMenu("Файл");
13      JMenuItem m;
14      m = new JMenuItem("Новый"); myMenu.add(m);
15      m=new JMenuItem("Открыть"); myMenu.add(m);
16      m = new JMenuItem("Сохранить");
17      myMenu.add(m);
18      m = new JMenuItem("Выход"); myMenu.add(m);
19      JMenuBar myMenuBar = new JMenuBar();
20      myMenuBar.add(myMenu);
21      setJMenuBar(myMenuBar);
22      }
23  }
```

В конструкторе класса `MenuDemo` (строки с 8 по 22) последовательно выполняются следующие действия:

– создание пунктов меню «Файл»: «Новый» (строка 14), «Открыть» (строка 15), «Сохранить» (строки 16, 17), «Выход» (строка 18);

– создание панели для главного меню `myMenuBar` (строка 19) и поместили в нее меню «Файл» (строка 20);

– задание панели главного меню (строка 21).

Результат работы программы приведен на рисунке 3.1.

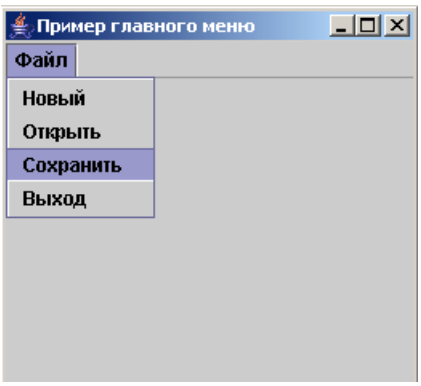


Рисунок 3.1 — Главное меню программы

3.2.2 Вложенные меню

Класс `JMenu` является потомком класса `JMenuItem`. Следовательно, каждый объект класса `JMenu` является одновременно объектом класса `JMenuItem` и может быть элементом меню в другом меню. Ниже приведен пример исходного кода, иллюстрирующий создание нескольких вложенных меню (рисунок 3.2):

```
import javax.swing.*;
public class SubMenus extends JFrame {
    public static final int WIDTH = 600;
    public static final int HEIGHT = 300;
    public SubMenus() {
        setSize(WIDTH, HEIGHT);
        addWindowListener(new WindowDestroyer());
        setTitle("Вложенные меню");
        JMenu mainMenu = new JMenu("Файл");
        JMenuItem m;
        JMenu saveMenu = new JMenu("Сохранить");
        m = new JMenuItem("В папке 1");
        saveMenu.add(m);
        m = new JMenuItem("В папке 2");
        saveMenu.add(m);
        mainMenu.add(saveMenu);
        JMenu getMenu = new JMenu("Открыть");
        m=new JMenuItem("Из папки Мои документы");
        getMenu.add(m);
        m = new JMenuItem("Из папки проекта");
        getMenu.add(m);
        mainMenu.add(getMenu);
        m = new JMenuItem("Сброс");
        mainMenu.add(m);
        m = new JMenuItem("Выход");
        mainMenu.add(m);
        JMenuBar mBar = new JMenuBar();
        mBar.add(mainMenu);
        setJMenuBar(mBar);
```

```
    }
    public static void main(String[] args) {
        SubMenus sm = new SubMenus();
        sm.setVisible(true);
    }
}
```

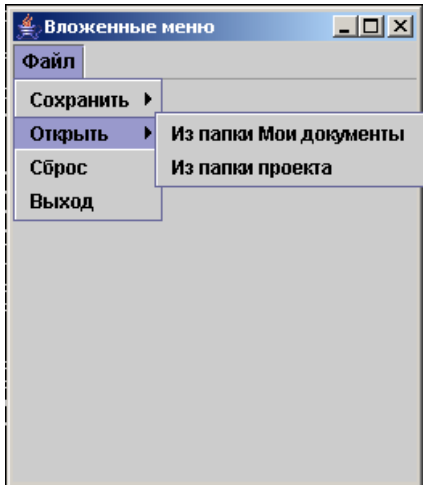


Рисунок 3.2 — Пример вложенного меню

3.2.3 Добавление пиктограмм в пункты меню

Меню выглядит гораздо лучше, если его команды иллюстрированы пиктограммами, привычными для пользователя. Для компоновки пиктограмм в пунктах меню используются объекты класса `ImageIcon`. Рассмотрим пример:

```
1   import javax.swing.*;
2   public class MenuIcon extends JFrame
3   {
4       public static final int WIDTH = 600;
5       public static final int HEIGHT = 300;
6       public static void main(String[] args) {
7           MenuIcon mi= new MenuIcon();
8           mi.setVisible(true); }
9       public MenuIcon()
10  {
11      addWindowListener(new WindowDestroyer());
12      setSize(WIDTH, HEIGHT);
```

```
13      setTitle("Пиктограммы в меню");
14      JMenu myMenu = new JMenu("Файл");
15      JMenuItem m;
16      m = new JMenuItem("Новый"); myMenu.add(m);
17      ImageIcon openIcon = new
    ImageIcon("open.gif");
18      m = new JMenuItem("Открыть");
19      m.setIcon(openIcon);
20      ImageIcon saveIcon = new
    ImageIcon("save.gif");
21      myMenu.add(m);
22      m = new JMenuItem("Сохранить");
23      m.setIcon(saveIcon); myMenu.add(m);
24      m = new JMenuItem("Выход"); myMenu.add(m);
25      JMenuBar myMenuBar = new JMenuBar();
26      myMenuBar.add(myMenu);
27      setJMenuBar(myMenuBar);
28      }
29  }
```

В приведенном примере в строках 17 и 20 создаются объекты класса `ImageIcon`, в которые загружаются соответствующие пиктограммы. Объекты `ImageIcon` подключаются к пунктам меню в строках 19 и 23 методом `setIcon`.

Результат работы программы показан на рисунке 3.3.

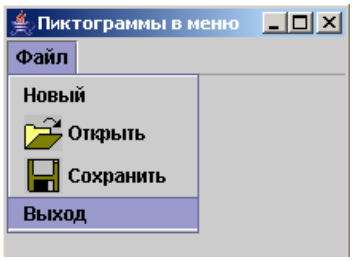


Рисунок 3.3 — Главное меню с пиктограммами

3.3 Класс контекстного меню JPopupMenu

Меню, которые появляются в программе при щелчке правой клавишей мыши, называют всплывающими (popup menu), или

контекстными. Всплывающие меню в Swing реализованы классом JPopupMenu. Для создания пунктов меню используются объекты класса JMenuItem.

3.3.1 Создание контекстного меню

Последовательность создания контекстного меню такая же, как и главного меню. Ниже приведен пример с соответствующими комментариями. На рисунке 3.4 показан результат работы программы при нажатии правой клавиши мыши в пределах формы.

```
import javax.swing.*;
import java.awt.event.*;
public class PopupMenus extends JFrame {
public PopupMenus() {
super("PopupMenus");
setDefaultCloseOperation(EXIT_ON_CLOSE);
// создаем контекстное меню
popup = createPopupMenu();
// назначаем для панели
// слушателя событий от мыши
addMouseListener(new ML());
// Установка размера и вывод окна на экран
setSize(300, 200); setVisible(true);
}
// Создание контекстного меню
private JPopupMenu createPopupMenu() {
// создаем само контекстное меню
JPopupMenu pm = new JPopupMenu();
// создаем пункты меню
JMenuItem first = new JMenuItem("Первый");
JMenuItem second = new JMenuItem("Второй");
JMenuItem third = new JMenuItem("Третий");
// и добавляем их методом add()
pm.add(first);pm.add(second); pm.add(third);
return pm; }
private JPopupMenu popup;
// Класс для обработки щелчка мыши
```

```
class ML extends MouseAdapter {
public void mouseClicked(MouseEvent me) {
/* проверяем, что нажата правая кнопка, если да,
выводим меню в соотв. позицию */
if (SwingUtilities.isRightMouseButton(me)) {
popup.show(getContentPane(), me.getX(), me.getY());
}
} }
public static void main(String[] args)
{ new PopupMenus(); }
}
```

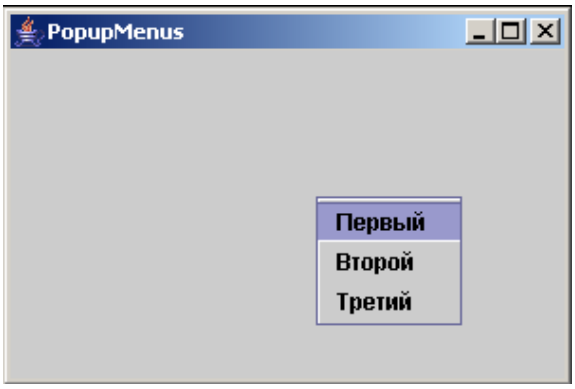


Рисунок 3.4 — Контекстное меню

Сравните последний пример с предыдущим и найдите отличия и общее в коде создания контекстного и главного меню.

3.3.2 Пиктограммы и вложенные меню в контекстном меню

Добавление вложенных пунктов в контекстное меню, а также прикрепление пиктограмм к командам меню производится так же, как и для главного меню (см. п. 3.2.2 и п. 3.2.3).

3.4 Класс панели инструментов JToolBar

Панели инструментов предназначены для запуска наиболее часто используемых команд приложения посредством набора кнопок (как правило, без надписей, но с подсказками и с небольшими четко различимыми значками). В панелях инструментов также встречаются наиболее востребованные пользователями компоненты, находить которые в меню или диалоговых окнах долго и неудобно. В Swing

панели инструментов представлены классом JToolBar. С его помощью можно создавать любые панели инструментов.

Для конструирования панели инструментов следует создать объект класса JToolBar, добавить в него кнопки или другие компоненты (особенно удобно использовать для панелей инструментов объекты-«команды» Action, в которых сразу можно указать и параметры внешнего вида кнопки, и описать то, что должно происходить при щелчке на ней) и вывести панель инструментов на экран. Рассмотрим пример.

```
1   import javax.swing.*;
2   import java.awt.event.*;
3   public class SimpleToolbars extends JFrame {
4   public SimpleToolbars() {
5   super("SimpleToolbars");
6   setDefaultCloseOperation(EXIT_ON_CLOSE);
7   // первая панель инструментов – объект
8   JToolBar toolbar1 = new JToolBar();
9   // добавление кнопок
10  toolbar1.add(new JButton(
11  new ImageIcon("images/New16.gif")));
12  toolbar1.add(new JButton(
13  new ImageIcon("images/Open16.gif")));
14  // разделитель между кнопками
15  toolbar1.addSeparator();
16  // добавляем команду
17  toolbar1.add(new SaveAction());
18  // вторая панель инструментов
19  JToolBar toolbar2 = new JToolBar();
20  // добавляем команду
21  toolbar2.add(new SaveAction());
22  // добавляем раскрывающийся список
23   toolbar2.add(new JComboBox(new String[] {
24   "Жирный", "Обычный" }));
25  // добавим панели инструментов в окно
26  getContentPane().add(toolbar1, "North");
```

```
27  getContentPane().add(toolbar2, "South");
28  // выводим окно на экран
29  setSize(400, 300); setVisible(true);
30  }
31  // команда для панели инструментов
32  class SaveAction extends AbstractAction {
33  public SaveAction() {
34  // настроим значок команды
35  putValue(AbstractAction.SMALL_ICON,
36  new ImageIcon("images/Save16.gif"));
37  // текст подсказки
38  putValue(AbstractAction.SHORT_DESCRIPTION,
29  "Сохранить документ..."); }
40  public void actionPerformed(ActionEvent e) {
41  // ничего не делаем
42  }
43  }
44  public static void main(String[] args) {
45  new SimpleToolbars();
46  }
47  }
```

В программе создаются две панели инструментов. Вначале демонстрируется наиболее распространенный способ использования панели инструментов: создается объект `JToolBar`, в него добавляются кнопки `JButton` с пиктограммой. После двух кнопок добавляется разделитель методом `addSeparator()`. Третья кнопка добавляется не в виде объекта `JButton`, а как экземпляр команды `Action`, добавить команду позволяет специальная перегруженная версия метода `add()`. Команды `Action` во многих случаях очень удобны, так как позволяют совмещать настройку внешнего вида элемента управления и описание действия, которое он выполняет. Это особенно верно для панелей инструментов: в классе команды задается значок и текст подсказки и тут же можно описать действие, которое должна будет выполнить команда. После этого нужно добавить команду в панель инструментов.

Если же необходимо модифицировать команду, будь это ее внешний вид или действие, то все скрыто в одном классе.

Вторая панель инструментов демонстрирует, что храниться в ней могут не только кнопки, но и любые другие компоненты. Сначала в панель добавляется команда, а затем раскрывающийся список `JComboBox`, созданный на основе массива строк.

Созданные панели инструментов добавляются в форму. Первая панель размещается на севере, а вторая — на юге главного окна. Стоит обратить внимание на специальные полосы с левого края панели инструментов (так называемые вешки перемещения): с их помощью можно перетаскивать (drag and drop) панели инструментов, прикреплять (drag and dock) их к разным края окна, или же оставлять «плавающими».

Результат работы программы приведен на рисунке 3.5.

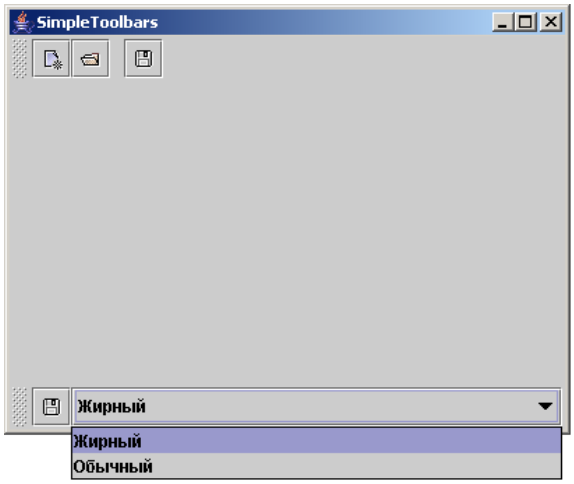


Рисунок 3.5 — Панели инструментов

В таблице 3.1 перечислены свойства панелей инструментов `JToolBar`, позволяющие настраивать их поведение.

Таблица 3.1 — Основные свойства панелей инструментов

Свойство	Описание
orientation	Задает направление, в котором будут добавляться компоненты на панель инструментов: вертикальное или горизонтальное (по умолчанию)
floatable	Управляет возможностью перетаскивания панели инструментов (по умолчанию включено)

Продолжение таблицы 3.1

Свойство	Описание
rollover	Включает эффект интерактивности — при наведении на компонент указателя мыши у него появляется специальная рамка или особый значок. По умолчанию этот эффект отключен. Работает эффект только для кнопок (наследников класса AbstractButton)
borderPainted	Управляет прорисовкой рамки панели инструментов. Наличие рамки зависит от внешнего вида и поведения, и по умолчанию рамка отображается. Как правило, это специальная полоса, с помощью которой пользователь может перетаскивать панель инструментов

3.5 Диалоги JFileChooser и JColorChooser

Компонент `JFileChooser` предназначен для выбора файлов и каталогов. Особенности различных файловых систем скрыты в наследниках абстрактного класса `FileSystemView`, поэтому выбранный для приложения внешний вид отобразит файловую структуру соответственно операционной системе. Так как `JFileChooser` — это наследник класса `JComponent`, его можно включить в любое место интерфейса. Однако удобнее использовать методы, показывающие компонент для выбора файлов в собственном модальном диалоговом окне. Рассмотрим пример:

```
1   import javax.swing.*;
2   import java.awt.event.*;
3   public class SimpleFileChooser extends JFrame
4   { // создаем общий экземпляр JFileChooser
5   JFileChooser fc = new JFileChooser();
6   public SimpleFileChooser() {
7   super("SimpleFileChooser");
8   setDefaultCloseOperation(EXIT_ON_CLOSE);
9   // кнопка открытия диалогового окна
10  JButton open = new JButton("Открыть...");
11  open.addActionListener(new ActionListener() {
12  public void actionPerformed(ActionEvent e) {
```

```
13  fc.setDialogTitle("Выберите каталог");
14  // Выбор каталога
15  fc.setFileSelectionMode(
    JFileChooser.DIRECTORIES_ONLY);
17  int res = fc.showOpenDialog
     (SimpleFileChooser.this);
19  // Если файл выбран, отметим это
20  if ( res == JFileChooser.APPROVE_OPTION )
21  JOptionPane.showMessageDialog(
22  SimpleFileChooser.this,fc.getSelectedFile());}
26  }); // кнопка открытия диалогового окна
28  JButton save = new JButton("Сохранить...");
29  save.addActionListener(new ActionListener() {
30  public void actionPerformed(ActionEvent e) {
31  fc.setDialogTitle("Сохранение файла");
32  // настройка режима
33  fc.setFileSelectionMode(
34  JFileChooser.FILES_ONLY);
35  int res = fc.showSaveDialog(
36  SimpleFileChooser.this);
37  // Выбор сделан
38  if ( res == JFileChooser.APPROVE_OPTION )
39  JOptionPane.showMessageDialog(
40  SimpleFileChooser.this, "Файл сохранен");}});
41  // добавим кнопки и выведем окно на экран
42  JPanel contents = new JPanel();
43  contents.add(open); contents.add(save);
44  setContentPane(contents);setSize(300, 200);
45  setVisible(true);                          }
46  public static void main(String[] args) {
47  new SimpleFileChooser();                    }
48  }
```

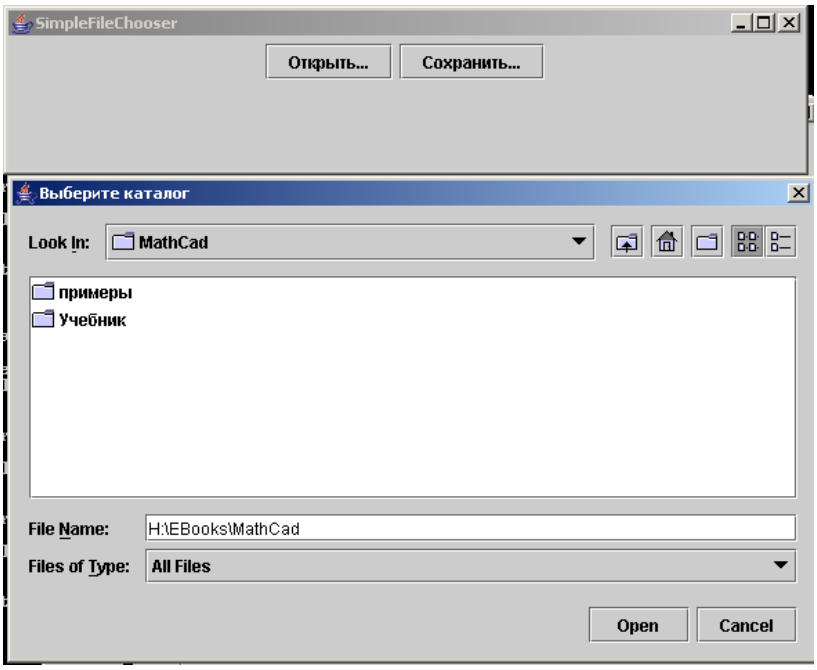


Рисунок 3.6 — Диалог для выбора файлов и каталогов

В примере создается окно с двумя кнопками, после нажатия на которые появляются диалоговые окна для открытия и сохранения файлов. В программе используется один экземпляр объекта для выбора файлов `JFileChooser`, как для сохранения файлов, так и для их открытия. Заголовок для диалогового окна можно задать методом `setDialogTitle()`. Перед выводом диалогового окна для выбора файлов на экран настраивается режим выбора. Компонент `JFileChooser` может работать в одном из трех режимов (режим выбора хранится в свойстве `fileSelectionMode`), перечисленных в таблице 3.2.

Таблица 3.2 — Режимы работы JFileChooser

Режим работы	Описание
FILES_ONLY	Пользователю для выбора (независимо от того, сохраняется файл или открывается) будут доступны только файлы, но не каталоги. По умолчанию JFileChooser работает именно в этом режиме.
FILES_AND_DIRECTORIES	В этом режиме пользователь может выбирать и каталоги, и файлы.

Продолжение таблицы 3.2

Режим работы	Описание
DIRECTORIES_ONLY	Этот режим разрешает пользователю выбирать исключительно каталоги.

Выше был выбран последний режим, поэтому открывать можно только каталоги. Для вывода диалогового окна на экран служит метод `showOpenDialog()`, которому нужно передать ссылку на родительский компонент, относительно которого окно будет располагаться на экране. В ответ метод возвращает константу, определяющую сделанный пользователем выбор (табл. 3.3).

Таблица 3.3 — Значения, возвращаемые методами класса JFileChooser

Константа	Описание
APPROVE_OPTION	Выбор файла в диалоговом окне прошел успешно, можно получить выбранный файл методом getFile()
CANCEL_OPTION	Пользователь отменил выбор файла, щелкнув на кнопке Cancel
ERROR_OPTION	При выборе файла произошла ошибка, или пользователь закрыл диалоговое окно для выбора файлов.

В примере выше происходит проверка: успешен ли был выбор файла, и если да (метод `showOpenDialog()` вернул значение `APPROVE_OPTION`), выводится на экран имя выбранного каталога (с помощью класса `JOptionPane`).

При нажатии на вторую кнопку появляется диалоговое окно сохранения файлов. Если выбор файла для сохранения проходит успешно (диалог возвращает значение `APPROVE_OPTION`), на экране появляется краткое сообщение, подтверждающее успешное сохранение файла.

Компонент `JFileChooser` предназначен всего лишь для выбора файла, а проводить с выбранным файлом соответствующие манипуляции (записывать в него информацию или анализировать его

содержимое) необходимо самостоятельно, написав соответствующий программный код.

3.6 Фильтры файлов

По умолчанию в списке файлов, который компонент `JFileChooser` предоставляет пользователю, содержатся все файлы из текущего каталога. Однако чаще всего известно, что пользователь, скорее всего, откроет файл с определенным именем или типом. Можно сократить круг поиска пользователя с помощью фильтра файлов, который встраивается в компонент `JFileChooser`.

Фильтры файлов для `JFileChooser` описаны в абстрактном классе `FileFilter` (пакет `javax.swing.filechooser`). Для создания фильтра нужно наследовать класс `FileFilter`, основным методом которого является `accept()`. Этот метод вызывает класс `JFileChooser` для каждого файла, перед тем как добавить его в список файлов, предоставляемых для выбора пользователю. В методе `accept()` необходимо проанализировать файл и решить, выводить его в списке или нет. Рассмотрим пример.

```
1   import javax.swing.*;
2   import javax.swing.filechooser.*;
3   public class FilteringFiles extends JFrame {
4   public FilteringFiles() {
5   super("FilteringFiles");
6   setDefaultCloseOperation(EXIT_ON_CLOSE);
7   // выводим окно на экран
8   setSize(300, 200); setVisible(true);
9   // настраиваем компонент выбора файла
10  JFileChooser chooser = new JFileChooser();
11  chooser.setDialogTitle("Выберите    текстовый
    файл");
12  // присоединяем фильтр
13  Chooser.addChoosableFileFilter(
14  new TextFilesFilter());
15  // выводим диалоговое окно на экран
16  int res = chooser.showOpenDialog(this);
```

```
17  if (res == JFileChooser.APPROVE_OPTION)
18   JOptionPane.showMessageDialog(this,
19  chooser.getSelectedFile());
20  }
21  // фильтр, отбирающий текстовые файлы
22  class TextFilesFilter extends FileFilter {
23  // принимает файл или отказывает ему
24  public boolean accept(java.io.File file) {
25  // все каталоги принимаем
26  if ( file.isDirectory() ) return true;
27  // для файлов смотрим на расширение
28  return (file.getName().endsWith("txt") );
29  }
30  // метод возвращает описание фильтра
31  public String getDescription() {      return
    "Текстовые файлы (*.txt)";
32  }
33  }
34  public static void main(String[] args) {
35  new FilteringFiles();      }
36  }
```

Фильтр файлов создается в строках 22-33 путем наследования внутреннего класса `TextFilesFilter` от абстрактного класса `FileFilter` и переопределения двух методов базового класса. Метод `getDescription()` должен сообщать краткое описание фильтра файлов. Обычно в это описание добавляется правило, по которому действует фильтр. В случае выше указана маска, по которой отбираются файлы. В методе `accept()` проводится анализ имени файла и оценка его пригодности для включения в список файлов, который будет показан пользователю. Стоит обратить внимание, что в данный метод попадают все файлы и каталоги, поэтому происходит проверка: не каталог ли попался, а все каталоги включаются в список (в противном случае, при отсутствии в списке каталогов, пользователь просто не сможет перемещаться по файловой системе). Для обычных файлов

проводится анализ имени: наш фильтр должен отбирать текстовые файлы, а их имена (точнее расширения) заканчиваются символами «txt».

Присоединить фильтр файлов к объекту `JFileChooser` можно двумя способами. В примере используется вызов метода `addChoosableFileFilter()`, который присоединит фильтр файлов к списку остальных фильтров (а в компоненте `JFileChooser` всегда имеется один фильтр — тот, что пропускает все файлы и каталоги). Есть и второй способ — вызов метода `setFileFilter()`.

Запустив программу с примером, Вы увидите, что после присоединения фильтра файлов компонент `JFileChooser` стал отображать только текстовые файлы. Описание, переданное из метода `getDescription()`, появляется в раскрывающемся списке доступных фильтров файлов. Фильтры файлов намного облегчают жизнь пользователя, особенно при выборе из большого количества файлов. Результат работы программы представлен на рисунке 3.7.

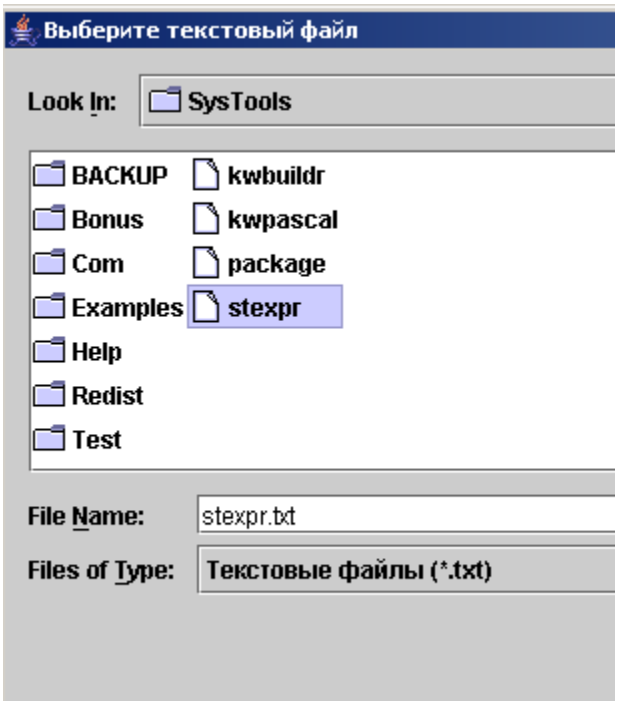


Рисунок 3.7 — Фрагмент окна выбора файла с фильтром

Еще одно стандартное диалоговое окно, реализовано в классе `JColorChooser` и предназначено для быстрого и удобного выбора цвета. Типичная последовательность выбора цвета такова: создается

объект класса JColorChooser и по мере необходимости происходит вывод его на экран в диалоговом окне стандартного вида с помощью метода showDialog(), попутно задавая родительский компонент и заголовок окна. Рассмотрим пример.

```
1   import javax.swing.*;
2   import java.awt.Color;
3   import java.awt.event.*;
4   public class SimpleColorChooser extends
    JFrame {
5   // Панель содержимого
6   private JPanel contents = new JPanel();
7   // компонент для выбора цвета
8   private JColorChooser chooser =
                        new JColorChooser();
9   public SimpleColorChooser() {
10  super("SimpleColorChooser");
11  setDefaultCloseOperation(EXIT_ON_CLOSE);
12  JButton choose = new JButton("Выбор цвета
    фона");
13  choose.addActionListener(new ActionListener()
    {
14  public void actionPerformed(ActionEvent e) {
15  Color color = chooser.showDialog(
16  SimpleColorChooser.this,
17  "Выберите цвет фона",
18  contents.getBackground());
19  // если цвет выбран, используем его
20  if ( color != null)
21  contents.setBackground(color);
22  repaint();
23  }
24  });
25  // выводим окно на экран
26  contents.add(choose);
27  setContentPane(contents);
```

```
28   setSize(300, 200); setVisible(true);
29   }
30   public static void main(String[] args) {
31   new SimpleColorChooser();         }
32   }
```

В данном примере компонент `JColorChooser` (строка 8) позволяет выбирать цвет фона для панели содержимого. В качестве панели содержимого выступает панель `JPanel` (строка 6), которая по умолчанию непрозрачна, то есть имеет цвет фона. Поэтому для смены цвета фона всего окна достаточно сменить цвет фона панели содержимого, вызвав метод `setBackground()`.

Объект `JColorChooser`, и `JFileChooser`, обычно создается в программе один раз. Это значительно ускоряет процесс появления диалогового окна на экране и экономит ресурсы. Вывести стандартное диалоговое окно для выбора цвета на экран просто: надо вызвать метод `showDialog()`, которому передается три параметра:

– родительский компонент диалогового окна (это может быть любой компонент, относительно него диалоговое окно располагается на экране);

– заголовок диалогового окна;

– цвет, изначально выбранный в компоненте `JColorChooser`.

Третий параметр можно задать пустым (`null`), в этом случае изначально цвет выбран не будет. В ответ метод `showDialog()` вернет цвет `Color`, который выбрал пользователь, или пустую (`null`) ссылку, если пользователь передумал и решил цвет не выбирать. В примере осуществляется проверка, что выбор был сделан, и устанавливается выбранный цвет в качестве цвета фона.

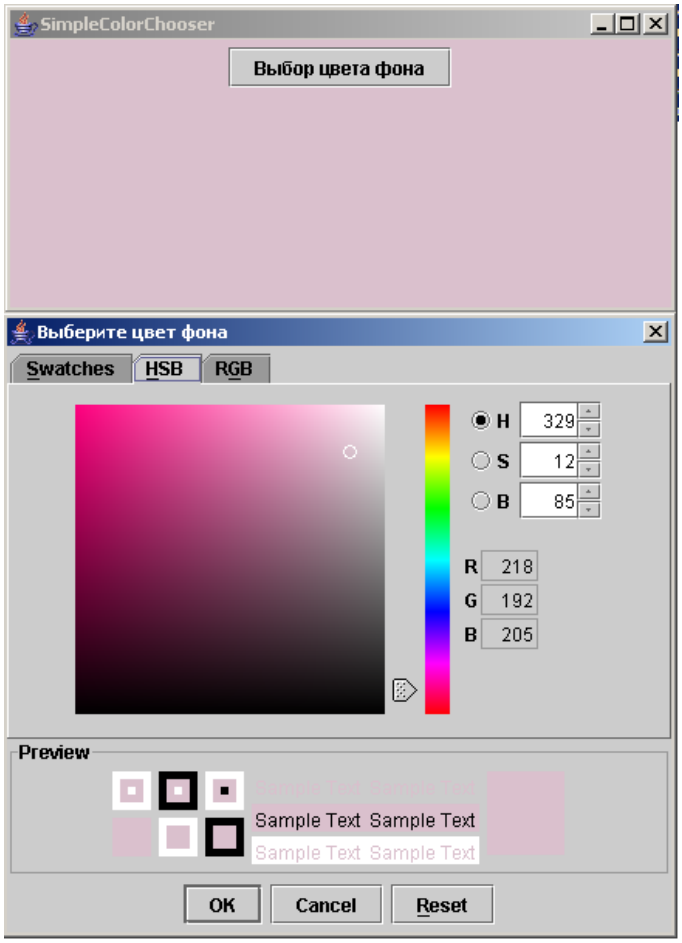


Рисунок 3.8 — Программа с диалогом выбора цвета

В компоненте `JColorChooser` имеется возможность дополнительной настройки: к трем стандартным способам выбора цвета (из заранее определенного набора, с помощью цветовой модели RGB или HSB) можно добавить собственную панель для выбора цвета, унаследовав ее от абстрактного класса `AbstractColorChooserPanel` из пакета `javax.swing.colorchooser`. Однако требуется это крайне редко — стандартные способы выбора цвета, представленные компонентом `JColorChooser`, достаточно полны. Кроме того, можно присоединить к компоненту `JColorChooser` панель просмотра (методом `setPreviewPanel()`), в которой можно показать, как смена цвета скажется на работе пользователя еще до того, как он окончательно сделает свой выбор. В панели просмотра можно показать, к примеру, уменьшенную копию изображения с новыми цветами. Для того чтобы узнать о смене цвета в компоненте `JColorChooser` еще до

окончательного выбора, имеется модель выбора цвета ColorSelectionModel. Присоединив к этой модели слушателя ChangeListener, можно узнать, какие цвета «опробует» пользователь перед тем, как принять окончательное решение.

3.7 Классы панелей JTabbedPane и JScrollPane

Панель с вкладками JTabbedPane позволяет выводить на экран набор панелей с небольшими ярлыками на краю (верхнем, нижнем, левом или правом). При щелчке на ярлыке панель JTabbedPane выводит на экран соответствующие выбранной вкладке элементы пользовательского интерфейса. Такой подход дает возможность сэкономить место в контейнере: вместо компоновки всех компонентов интерфейса в одной панели содержимого происходит их разделение (чаще всего по смыслу и предназначению) на несколько групп. Пользователь будет видеть только одну группу компонентов, а чтобы увидеть и воспользоваться другой группой, ему нужно перейти на подходящую вкладку. Чаще всего панели с вкладками используются в диалоговых окнах настройки приложения: в них можно разместить много компонентов с различным предназначением, а вкладки не только помогают компактно разместить эти компоненты, но и дают пользователю возможность быстро найти то, что его интересует.

Работа JTabbedPane заключается в вызове метода add() или addTab(), которому необходимо передать компонент, соответствующий вкладке, надпись для ярлычка вкладки и значок. После добавления всех вкладок панель JTabbedPane выводится на экран, при этом стоит проследить за тем, чтобы занимаемого ею места было достаточно не только для тех вкладок, которые были добавлены в нее, но и для соответствующих этим вкладкам компонентов. Рассмотрим пример.

```
1    import javax.swing.*;
2    import java.awt.GridLayout;
3    public class SimpleTabbedPanes extends JFrame {
4    public SimpleTabbedPanes() {
5    super("SimpleTabbedPanes");
6    setDefaultCloseOperation(EXIT_ON_CLOSE);
```

```
7   // первая панель с вкладками
8   JTabbedPane tabsOne = new JTabbedPane(
9   JTabbedPane.BOTTOM,
    JTabbedPane.SCROLL_TAB_LAYOUT);
10  // добавление вкладок
11  for (int i=1; i<8; i++) {
12  JPanel tab = new JPanel();
13  tab.add(new JButton("Просто кнопка "+ i));
14  tabsOne.addTab("Вкладка №: " + i, tab);
15  }
16  // вторая панель с вкладками
17  JTabbedPane tabsTwo = new
        JTabbedPane(JTabbedPane.TOP);
18  // добавление вкладок
19  for (int i=1; i<8; i++) {
20  JPanel tab = new JPanel();
21  tab.add(new JButton("Снова кнопка " + i));
22  tabsTwo.addTab("<html><i>Вкладка №: " + i,
    new ImageIcon("icon.gif"), tab, "Нажмите " + i
    + "!");
23  }
24  // добавление вкладки в панель содержимого
25  getContentPane().setLayout(new GridLayout());
26  getContentPane().add(tabsOne);
27  getContentPane().add(tabsTwo);
28  // вывод окна на экран
29  setSize(600, 250); setVisible(true);
30  }
31  public static void main(String[] args) {
32  New SimpleTabbedPanes();          }
33  }
```

Результат работы программы представлен на рисунке 3.9.

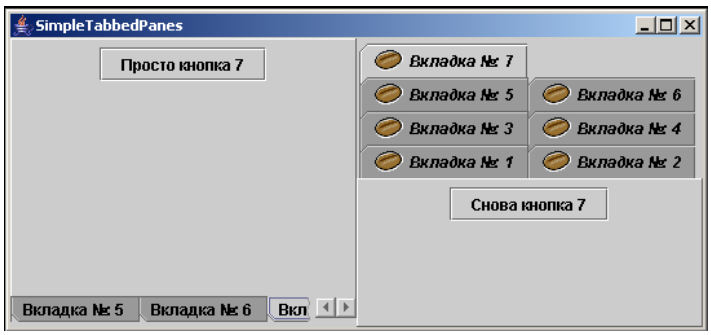


Рисунок 3.9 — Панель с вкладками

Панель прокрутки `JScrollPane` позволяет организовать прокрутку компонента, а также при необходимости дополнить ее другими возможностями. Заголовки (headers) панели прокрутки дополняют прокручиваемый компонент сверху и слева, предоставляя пользователю вспомогательную информацию о том содержимом, которое он «прокручивает». Как правило, заголовки используются в качестве навигационных или информационных дополнений: линеек или координатных осей, которые помогают быстро определить размеры прикручиваемой области, текущую позицию или получить иную вспомогательную информацию. Заголовки находятся в «видоискателях» `JViewport`, так же как и основное содержимое панели прокрутки. Это позволяет им иметь большой размер, как правило, совпадающий с размером области прокрутки (что весьма логично: заголовки должны предоставлять вспомогательную информацию обо всей прокручиваемой области). Если позиция прокручиваемого компонента изменяется пользователем с помощью полос прокрутки, то позиция заголовка меняется панелью прокрутки `JScrollPane` автоматически, синхронно с позицией основного содержимого.

Уголки (corners) панели прокрутки реже применяются в приложениях. Это обычные компоненты, которые занимают пустые места по углам панели прокрутки. Этих пустых мест может и не быть – они появляются только при наличии полос прокрутки и заголовков. В уголки обычно добавляются компоненты, которые выполняют масштабирование прокручиваемого изображения, переход на позицию по некоторому признаку или вызов контекстного меню с подходящим списком команд. Рассмотрим пример.

```
1   import javax.swing.*;
2   import java.awt.Color;
3   public class SimpleScrollPanes extends JFrame
    {
4   public SimpleScrollPanes() {
5   super("SimpleScrollPanes");
6   setDefaultCloseOperation(EXIT_ON_CLOSE);
7   // надпись
8   JLabel label = new JLabel(new
    ImageIcon("image.jpg"));
9   // Конструктор панели прокрутки
10  JScrollPane scrollPane = new
    JScrollPane(label,
11  JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
12  JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
13  // Свойства панели
14  scrollPane.setViewportBorder(
15  BorderFactory.createLineBorder(Color.blue));
16  scrollPane.setWheelScrollingEnabled(false);
17  // выводим окно на экран
18  getContentPane().add(scrollPane);
19  setSize(400, 300); setVisible(true);
20  }
21  public static void main(String[] args) {
22  New SimpleScrollPanes();       }
23  }
```

Результат работы программы представлен на рисунке 3.10.

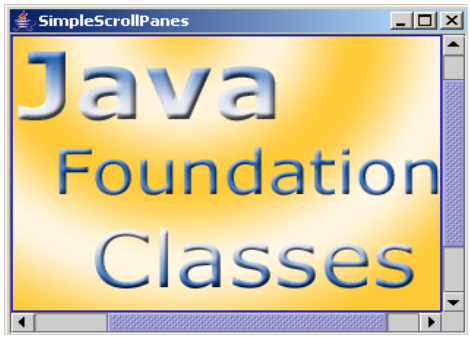


Рисунок 3.10 — Окно с панелью прокрутки

3.8 Задание к лабораторной работе

Создайте текстовой редактор на основе элемента управления `JTextPane` (см. Приложение Б):

1. Создайте программу — приложение Java на основе класса `JFrame`. Добавьте в окно главное меню, включающее пункты и команды согласно таблице 3.4.

Таблица 3.4 — Пункты главного меню

Пункт меню	Команда	Назначение	Иконка
Файл	Новый	Создание нового документа	
	Открыть	Загрузка документа с диска	
	Сохранить	Запись документа на диск под старым именем	
	Сохранить как…	Запись документа на диск с новым именем	
	Печать	Печать документа	
	Выход	Выход из программы	
Правка	Отменить	Отмена предыдущего действия	
	Повторить	Повтор отмененного действия	
	Вырезать	Забрать фрагмент текста в буфер обмена	
	Копировать	Скопировать фрагмент текста в буфер обмена	
	Вставить	Вставить фрагмент текста из буфера обмена	
	Удалить	Удалить фрагмент текста без помещения в буфер обмена	
	Выделить все	Выделить весь текст	
Формат	Шрифт	Установка параметров шрифта	
	Выравнивание	Выбор типа выравнивания в абзаце	
	Цвет	Цвет шрифта в выделенном фрагменте текста	
Справка	О программе	Вывод информации об авторе	

2. Программа должна обеспечивать следующие возможности:

– при закрытии, если в окне имеется документ, запросить подтверждение на сохранение;

– при создании нового документа, если в окне имеется документ, запросить подтверждение на сохранение;

– обеспечивать фильтрацию расширений при открытии и сохранении файлов (см. Приложение В);

– давать пользователю возможность изменять параметры фрагментов текста: шрифт, цвет, выравнивание;

– обеспечивать «прокрутку» текста при выходе за границы окна.

3. Запустите программу:

– запишите свои данные (ФИО, группа, факультет) в окне редактора. Сохраните содержимое окна в файле ФИО.rtf;

– запишите данные о работе (название, цель) в окне редактора. Сохраните содержимое окна в файле ЛР.rtf;

– оформите абзацы текста различными шрифтами.

4. Распечатайте текст программы и внешний вид окна редактора.

3.9 Контрольные вопросы

1. Критерии качества пользовательского интерфейса. Перечислите и разъясните основные положения.

2. Последовательность взаимодействия пользователя с системой. Перечислите основные шаги и поясните их.

3. Как минимизировать действия пользователя при работе с мышью?

4. Правила открытия диалоговых окон в приложении.

5. Преимущества и недостатки использования клавиатуры при управлении работой приложения.

6. Как добавить главное меню в программу?

7. Как добавить контекстное меню в программу?

8. Как создать пункты главного (контекстного) меню?

9. Как создать вложенные меню?

10. Как добавить пиктограммы в пункты меню?

11. Как создать панель инструментов?

12. Как добавить кнопку в панель инструментов? Как задать на кнопке надпись? Как разместить на кнопке пиктограмму?

13. Как создать разделитель между кнопками в панели инструментов?

14. Свойства панели инструментов.

15. Как создать диалог для работы с файлами?

16. Как настроить диалог для работы с фалами на работу только с файлами?

17. Как настроить диалог для работы с фалами на работу с каталогами и файлами?

18. Режимы работы диалога JFileChooser.

19. Значения, которые возвращает JFileChooser при взаимодействии с пользователем.

20. Как создать фильтр для вывода файлов с определенными расширениями?

21. Как создать диалоговое окно выбора цвета? Как задать цвет по умолчанию?

4 ЛАБОРАТОРНАЯ РАБОТА №4

Тема работы: сервлеты и JSP (Java Server Pages).

Цель работы: научиться разрабатывать серверные приложения Java: сервлеты и серверные страницы сценариев.

4.1 Сервлеты — общие сведения

Сервлет (servlet) — это программа на языке Java, которая выполняется на стороне Web-сервера. Каждый сервлет реализует класс javax.servlet или javax.servlet.http.HTTPServlet. Web-сервер с поддержкой сервлетов может динамически воздействовать на Web-страницы, изменяя их содержимое.

4.1.1 Типовая структура сервлета

В примере ниже приведена типовая структура сервлета с соответствующими пояснениями.

```
1  import java.io.*;
2  import javax.servlet.*;
3  import javax.servlet.http.*;
4  public class SomeServlet extends HttpServlet {
5  public void doGet(HttpServletRequest request,
   HttpServletResponse response)
6  throws ServletException, IOException {
   // request позволяет прочитать настройки
   // и данные, полученные от HTML-формы
   //(например, введенные пользователем)
   // response определяет текст для HTTP-
   //страницы
7  PrintWriter out = response.getWriter();
   // out помещает контекст в браузер
8    }
9  }
```

Сервлет должен переопределять методы doGet (строка 5) или doPost, или оба эти метода, в зависимости от того, как была сформирована посылка формы на Web-странице. Эти методы имеют два параметра: HttpServletRequest и HttpServletResponse.

Класс HttpServletRequest содержит методы, позволяющие

прочитать данные, полученные от формы на Web-странице. Класс HttpServletResponse содержит методы, позволяющие вывести ответ пользователю на Web-страницу.

Объект PrintWriter (строка 7) позволяет выводить ответ для клиента.

4.1.2 Первый сервлет

Ниже приведен пример кода сервлета, выводящего строку «Hello, World» на Web-страницу.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloWorld extends HttpServlet {
       public void doGet(HttpServletRequest
                         request,
                         HttpServletResponse
                         response)
throws ServletException, IOException {
    PrintWriter out = response.getWriter();
    out.println("Hello World");
    }
}
```

В данном примере используется только один метод объекта PrintWriter — println. Он выводит строку в контекст — на Web-страницу, которую получит клиент. Класс PrintWriter определен в пакете java.io, поэтому его обязательно нужно импортировать.

4.1.3 Жизненный цикл сервлета

Когда сервлет запускается, Web-сервер вызывает метод init(). Для обработки данных, полученных от HTML-формы, сервер вызывает метод service(). Для завершения работы сервлета необходимо использовать метод destroy().

Порядок работы при использовании сервлета выглядит так:

1. Пользователь вводит адрес в строке браузера. Браузер формирует запрос и посылает его соответствующему серверу.

2. Сервер, получив запрос, перенаправляет его специальному

сервлету, который загружается в адресное пространство сервера.

3. Сервер вызывает метод `init()` сервлета и запускает его.

4. Сервер с помощью метода `service()` сервлета обрабатывает HTTP-запрос. Сервлет может обработать полученные данные и сформировать ответ для клиента. Далее сервлет остается в памяти и реагирует на получаемые запросы, используя метод `service()`.

5. Сервер завершает работу сервлета путем вызова метода `destroy()`. При этом важные данные могут быть сохранены в постоянном хранилище.

4.1.4 Установка и запуск сервлетов

Для запуска сервлетов необходимо установить:

– Web-сервер;

– контейнер сервлетов;

– файлы классов для Servlet API;

– дескриптор развертывания.

В качестве Web-сервера можно использовать:

– службу Internet Information Services (IIS), входящую в состав Windows 2000/XP Professional/2003;

– Web-сервер Apache Tomcat, полностью написанный на Java;

– другие.

В качестве контейнера для сервлетов можно использовать множество программных продуктов, таких как:

– Jakarta фирмы Apache Tomcat (бесплатный продукт);

– ServletExec (надстройка для IIS) фирмы New Atlanta (коммерческий продукт);

– другие.

Совместное использование IIS и Apache Tomcat возможно, но нежелательно, т.к. требует дополнительных настроек операционной системы.

Поэтому в дальнейшем будем использовать надстройку ServletExec.

Рассмотрим установку надстройки (plug-in) ServletExec, необходимо:

1. Остановить работу IIS.

2. Запустить файл инсталляции ServletExec_ISAPI_50013.exe. Следовать инструкциям программы установки.

3. Перезапустить IIS.

4. Контейнер готов к работе.

Установка продуктов Apache подробно описана в [7].

Для того, чтобы сервлеты работали корректно классы следует помещать в папку `C:\InetPub\Scripts`. Вызов сервлета на выполнение осуществляется путем ввода в адресной строке браузера строки, имеющей вид:

`http://localhost/servlet/ИмяСервлета`

например

`http://localhost/servlet/TestServlet`.

4.1.5 Классы для создания сервлетов

Пакет `javax.servlet` содержит набор интерфейсов и классов, устанавливающих окружение, в котором работают сервлеты (таблица 4.1). Наиболее важный из них — это `Servlet`. Все сервлеты должны реализовывать указанный интерфейс или расширять класс, который его реализует. Интерфейсы `ServletRequest` и `ServletResponse` организуют обмен данными с клиентом.

Таблица 4.1 — Интерфейсы пакета javax.servlet

Интерфейс	Назначение
Servlet	Объявляет методы жизненного цикла сервлета
ServletConfig	Позволяет сервлетам получать параметры инициализации
ServletContext	Активизирует возможности сервлетов для регистрации событий и доступа к информации об их среде
ServletRequest	Используется для чтения данных из запроса клиента
ServletResponse	Используется для записи данных в ответ клиента
SingleThreadModel	Указывает, что сервлет защищен от многопоточности

В таблице 4.2 приведены классы, которые поддерживаются пакетом `javax.servlet`.

Таблица 4.2 — Классы пакета javax.servlet

Класс	Назначение
GenericServlet	Реализует интерфейсы Servlet и ServletConfig
ServletInputStream	Обеспечивает входной поток для чтения запросов от клиента
ServletOutputStream	Обеспечивает выходной поток для записи ответов клиенту
ServletException	Указывает, что произошла ошибка сервлета
UnavailableException	Указывает, что сервлет постоянно или временно недоступен

Интерфейс `Servlet` объявляет методы `init()`, `service()` и `destroy()`, которые вызываются сервером в течение жизни сервлета. Существует также метод, который позволяет сервлету получать любые параметры инициализации. Методы интерфейса `Servlet` приведены в таблице 4.3.

Таблица 4.3 — Методы интерфейса Servlet

Метод	Назначение
void destroy()	Вызывается, когда сервлет выгружается
ServletConfig getServletConfig()	Возвращает объект ServletConfig, который содержит все параметры инициализации
String getServletInfo()	Возвращает строку, описывающую сервлет
void init(ServletConfig sc) throws ServletException	Вызывается, когда сервлет инициализируется, sc – определяет параметры его инициализации. Если сервлет не может быть инициализирован, генерируется исключение UnavailableException
void service (ServletRequest req, ServletResponse res) throws ServletException, IOException	Вызывается, чтобы обработать запрос от клиента. Запрос от клиента может читаться из req. Ответ клиенту может быть записан в res. Если возникают проблемы в сервлете или при вводе/выводе, генерируется исключение ServletException или IOException

Интерфейс `ServletConfig` реализуется сервером. Он позволяет сервлету получать данные конфигурации (после своей загрузки). В таблице 4.4 описаны методы, объявленные в этом интерфейсе.

Таблица 4.4 — Методы интерфейса ServletConfig

Метод	Назначение
ServletContext getServletContext()	Возвращает контекст сервлета
String getInitParameter(String param)	Возвращает значение параметра инициализации с именем param
Enumeration getInitParameterNames()	Возвращает перечисление всех имен параметров инициализации

Интерфейс `ServletContext` реализуется сервером. Он позволяет сервлетам получать информацию об их среде выполнения. Описания его методов приведены в таблице 4.5.

Таблица 4.5 — Методы, определенные в ServletContext

Метод	Назначение
Object getAttribute(String attr)	Возвращает значение атрибута сервера с именем attr
String getMimeType (String file)	Возвращает MIME-тип файла
String getRealPath(String vpath)	Возвращает реальный путь, соответствующий виртуальному пути vpath
String getServerInfo ()	Возвращает информацию о сервере.

Интерфейс `ServletRequest` реализуется сервером. Он дает возможность сервлету получить информацию о запросе клиента. Описания его методов приводятся в таблице 4.6.

Таблица 4.6 — Методы, определенные в ServletRequest

Метод	Назначение
String getAttribute(String attr)	Возвращает значение атрибута с именем attr

Продолжение таблицы 4.6

Метод	Назначение
int getContentLength()	Возвращает размер запроса. Если размер неизвестен, возвращается значение –1
String getContentType()	Возвращает тип запроса. Если тип не может быть определен, возвращается значение null
ServletInputStream getlnputStream() throws IOException	Возвращает поток ServletInputStream, который можно использовать для чтения двоичных данных запроса. Если метод getReader() уже был вызван для этого запроса, генерируется исключение IllegalStateException
String getParamater(String pname)	Возвращает значение параметра с именем pname
Enumeration getParameterNames()	Возвращает перечисление имен параметров запроса
String[ ] getParameterValues()	Возвращает перечисление значений параметров запроса
String getProtocol()	Возвращает описание протокола
BufferedReader getReader() throws IOException	Возвращает буферизированное считывающее устройство, которое может быть использовано для чтения текста запроса. Если метод getInputStream() уже был вызван для этого запроса, генерируется исключение IllegalStateException
String getRealPath(String vpath)	Возвращает реальный путь, соответствующий виртуальному пути vpath
String getRemoteAddr()	Возвращает строковый эквивалент IP-адреса клиента
String getRemoteHost()	Возвращает имя хост-машины клиента
String getScheme()	Возвращает схему передачи URL, используемую для запроса (например "http", "ftp")
String getServerName()	Возвращает имя сервера
int getServerPort()	Возвращает номер порта сервера

Интерфейс `ServletResponse` реализуется сервером. Он дает возможность сервлету формировать ответ для клиента. Его методы описаны в таблице 4.7.

Таблица 4.7 — Методы, определенные в ServletResponse

Метод	Назначение
ServletOutputStream getOutputStream() throws IOException	Возвращает поток ServletOutputStream, который можно использовать для записи двоичных данных в ответ. Если метод getWriter() уже был вызван для этого запроса, выбрасывается исключение IllegalStateException
PrintWriter getWriter() throws IOException	Возвращает объект PrintWriter, который можно использовать для записи символьных данных в ответ. Если метод getOutputStream() уже был вызван для этого запроса, генерируется исключение IllegalStateException
void setContentLength(int size)	Устанавливает длину содержания ответа, равной значению size
void setContentType(String type)	Устанавливает тип содержания ответа согласно type

Интерфейс `SingleThreadModel` поддерживает однопоточное выполнение метода `service()` сервлета. Он не определяет никаких констант и не объявляет никаких методов. Если используется данный интерфейс, сервер создает несколько экземпляров сервлета. Когда поступает запрос клиента, он посылается доступному экземпляру сервлета.

Класс `GenericServlet` обеспечивает реализацию основных методов жизненного цикла сервлета. Разработчики сервлетов обычно работают с его подклассами. `GenericServlet` реализует интерфейсы `ServletConfig` и `Servlet`. Кроме того, имеется метод для добавления строки в файл журнала сервера. Сигнатура этого метода такова:

```
void log(String s),
```

где `s` — строка, которая добавляется в журнал регистрации.

Класс `ServletInputStream` расширяет класс `InputStream`. Он реализуется сервером и обеспечивает входной поток, который

можно использовать для чтения из запроса клиента. Кроме того, он содержит метод для чтения байтов из потока. Его сигнатура:

```
int readLine(byte[ ] buffer,
    int offset,
    int size) throws IOException,
```

где `buffer` — массив, в котором помещено `size` байтов, начиная с `offset`. Метод возвращает фактическое число считанных байтов, или «-1», если встретилось условие конца потока.

Класс `ServletOutputStream` расширяет класс `OutputStream`. Он также реализуется сервером и создает выходной поток, который можно использовать для записи данных в ответ клиенту. Он содержит методы `print()` и `println()`, которые собственно и выводят данные в поток.

Класс `ServletException` указывает, что в сервлете произошла какая-то проблема. Класс имеет следующие конструкторы: `SevletException()` и `ServletException(String s)`, где `s` — строка с описанием проблемы.

Класс `UnavailableException` расширяет класс `ServletException`. Он указывает, что сервлет постоянно или временно недоступен. Класс имеет следующие конструкторы:

```
UnavailableException(Servlet servlet, String s)
```

и

```
UnavailableException(int sees, Servlet servlet,
    String s)
```

Параметр `servlet` указывает, какой сервлет является недоступным. Описание проблемы указывается в параметре `s`. Число секунд, в течение которых сервлет, как ожидается, будет недоступным, указывается в `sees`.

4.1.6 Чтение параметров сервлета

Класс `ServletRequest` содержит методы, позволяющие читать имена и значения параметров, которые включены в запрос клиента. Рассмотрим пример сервлета, иллюстрирующего их использование. Пример содержит два файла: `PostParameters.htm` — Web-страница, вызывающая сервлет и

PostParametersServlet.java — собственно сервлет.

Web-страница определяет таблицу, которая содержит две текстовых метки и два текстовых поля. Одна из меток — Employee (Служащий), а вторая — Phone (Телефон). Форма также содержит кнопку для отправки данных в сервлет. Стоит обратить внимание, что параметр action тега формы <form ….> определяет адрес сервлета для обработки POST-запроса HTTP.

```
//файл PostParameters.htm
<html> <body> <center>
<form name="Forml" method="post"
action="http://localhost/servlet/
    PostParametersServlet">
<table> <tr>
    <td><B>Employee</td>
<td> <input type=textbox name="e" size="25"
value=""> </td> </tr> <tr>
    <td><B>Phone</td>
 <td><input type=textbox name="p" size="25"
value=""> </td> </tr>
</table>
<input type=submit value="Отправить">
</form> </body> </html>
```

Исходный код для PostParametersServlet.java показан в коде ниже. Метод service() переопределен для обработки запросов клиента. Метод getParameterNames() возвращает перечисление имен параметров. Они обрабатываются в цикле. Имя и значение параметра выводятся клиенту. Значение параметра возвращается с помощью метода getParameter().

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
public class PostParametersServlet
    extends GenericServlet {
public void service(ServletRequest request,
    ServletResponse response)
```

```
        throws ServletException, IOException {
    PrintWriter pw = response.getWriter();
    Enumeration e = request.getParameterNames();
    while(e.hasMoreElements()) {
    String pname = (String)e.nextElement();
    pw.print(pname + " = ");
    String pvalue = request.getParameter(pname);
    pw.println(pvalue);          } pw.close(); }    }
```

Откомпилируйте сервлет, поместите его класс в папку `C:\Inetpub\Scripts`. Отобразите Web-страницу в браузере. Введите в текстовые поля имя и телефонный номер служащего. Нажмите кнопку **Отправить** на Web-странице. После выполнения этих шагов, браузер отобразит ответ, который динамически генерируется сервлетом.

4.1.7 Пакет javax.servlet.http

Пакет `javax.servlet.http` содержит несколько интерфейсов и классов, которые обычно используются разработчиками сервлетов. Его возможности облегчают построение сервлетов, которые работают с HTTP-запросами и ответами. Таблица 4.8 описывает интерфейсы, которые определены в этом пакете, а таблица 4.9 — классы пакета.

Наиболее важный из классов — `HttpServlet`. Разработчики сервлетов обычно расширяют указанный класс, чтобы обработать запросы HTTP.

Таблица 4.8 — Интерфейсы пакета javax.servlet.http

Интерфейс	Назначение
HttpServletRequest	Разрешает сервлетам читать данные из HTTP-запроса
HttpServletResponse	Разрешает сервлетам записывать данные в HTTP-ответ
HttpSession	Позволяет читать и записывать данные сеанса
HttpSessionContext	Обеспечивает управляемость сеансов

Таблица 4.9 — Классы пакета javax.servlet.http

Класс	Описание
Cookie	Позволяет сохранять информацию состояния на машине клиента
HttpServlet	Обеспечивает методы для обработки запросов и ответов HTTP
UttpSessionBindingEvent	Указывает на наличие или отсутствие связи блока прослушивания с сеансовым значением
HttpUtils	Объявляет методы утилит для сервлетов

4.1.8 Интерфейс HttpServletRequest

Интерфейс `HttpServletRequest` реализуется сервером и позволяет сервлету получить информацию о запросе клиента. Его методы приведены в таблице 4.10.

Таблица 4.10 — Методы интерфейса HttpServletRequest

Интерфейс	Назначение
Cookie[] getCookies ()	Возвращает массив cookie-данных из запроса
long getDateHeader(String field)	Возвращает значение поля даты заголовка, передаваемого через параметр field
String getHeader(String field)	Возвращает значение поля заголовка, передаваемого через параметр field
Enumeration getHeaderNames()	Возвращает перечисление имен заголовка
String getMethod()	Возвращает HTTP-метод запроса
String getQueryString()	Возвращает строку запроса в URI
String getRemoteUser ()	Возвращает имя пользователя, который выдал этот запрос
String getRequestedSessionId()	Возвращает ID (идентификатор) сеанса
String getRequestURI() String	Возвращает часть URI слева от строки запроса
getServletPath()	Возвращает часть URI, которая идентифицирует сервлет

Продолжение таблицы 4.10

Интерфейс	Назначение
HttpSession getSession(boolean new)	Если new = true, то метод создает и возвращает новый сеанс для запроса. Иначе возвращает существующий сеанс этого запроса

4.1.9 Интерфейс HttpServletResponse

Интерфейс `HttpServletResponse` реализуется сервером. Он дает возможность сервлету формировать HTTP-ответ клиенту. В нем определено несколько констант, соответствующих различным кодам состояния, которые могут быть назначены HTTP-ответу. Например, `SC_OK` указывает, что HTTP-запрос достиг цели, а `SC_NOT_FOUND` указывает, что требуемый ресурс недоступен. Методы этого интерфейса кратко описаны в таблице 4.11.

Таблица 4.11 — Методы интерфейса HttpServletResponse

Интерфейс	Назначение
void addCookie(Cookie cookie)	Добавляет coocie-данные к HTTP-ответу
boolean containsHeader(String field)	Возвращает true, если HTTP-заголовок ответа содержит поле с именем field
String encodeURL(String url)	Определяет, нужно ли кодировать ID сеанса в URL-адресе, указанном в параметре url. Этим методом должны быть обработаны все URL-адреса, сгенерированные сервлетом
String encodeRedirectUrl(String url)	Определяет, нужно ли кодировать ID сеанса в URL-адресе, указанном в параметре url. Этим методом должны быть обработаны все URL-адреса, передаваемые в метод sendRedirect ()
void sendError(int c) throws IOException	Посылает клиенту код ошибки, указанный в параметре c
void sendError(int c, String s) throws IOException	Посылает клиенту код ошибки, указанный в параметре c, и сообщение, указанное в s
void sendRedirect(String url) throws IOException	Перенаправляет клиента к адресу, указанному в url

Продолжение таблицы 4.11

Интерфейс	Назначение
void setHeader(String field, String value)	Добавляет (в ответ) заголовок (в виде поля с именем, указанным в параметре field со значением, указанным в параметре value)
void setIntHeader(String field, int value)	Добавляет (в ответ) внутренний заголовок field, со значением value
void setStatus(int code)	Устанавливает код состояния ответа со значением, указанным в параметре code
void setStatus(int code, String s)	Устанавливает код состояния и сообщение данного ответа (со значениями, указанными в параметрах code и s)

4.1.10 Интерфейс HttpSession

Интерфейс `HttpSession` реализуется сервером. Он дает возможность сервлету прочитать и записать информацию о состоянии, которая связана с сеансом HTTP. Его методы кратко описаны в таблице 4.12. Все они генерируют исключение `IllegalStateException`, если сеанс был завершен.

Таблица 4.12 — Методы интерфейса HttpSession

Интерфейс	Назначение
long getCreationTime()	Возвращает время создания сеанса (в миллисекундах, начиная с полуночи 1 января 1970 г., GMT)
String getId()	Возвращает ID (идентификатор) сеанса
long getLastAccessedTime()	Возвращает время последнего запроса клиента в этом сеансе (в миллисекундах, начиная с полуночи 1 января 1970г., GMT)
HttpSessionContext getSessionContext()	Возвращает контекст, связанный с этим сеансом
Object getValue(String name)	Возвращает объект, связанный с именем, получаемым через параметр name. Возвращает null, если нет такой связи
String[] getValueNames()	Возвращает имена всех объектов, которые связаны с сеансом

Продолжение таблицы 4.12

Интерфейс	Назначение
void invalidate()	Завершает текущий сеанс и удаляет его из контекста
boolean isNewO	Возвращает true, если сервер создал сеанс, но он еще не был доступен клиенту
void putValue (string name, object obj)	Связывает объект obj с именем name в данном сеансе
void removeValue (String name)	Удаляет из сеанса объект, связанный с именем name

4.1.11 Интерфейс HttpSessionBindingListener

Интерфейс `HttpSessionBindingListener` реализуется объектами, которые нуждаются в уведомлении о наличии или отсутствии связи с сеансом HTTP. Методы, которые вызываются для установления наличия или отсутствия связи объекта с сеансом, имеют следующие сигнатуры:

```
void valueBound(HttpSessionBindingEvent e);
void valueUnbound(HttpSessionBindingEvent e),
```

где `e` — `event`-объект (объект события), который описывает событие, связанное с сеансом.

4.1.12 Интерфейс HttpSessionContext

Интерфейс `HttpSessionContext` реализуется сервером. Он дает возможность сервлету получить доступ к сеансу, который с ним связан. В нем объявлен метод, который возвращает перечисление всех сеансовых идентификаторов контекста, сигнатура данного метода: `Enumeration getIDs()`.

Другой метод может отображать `ID` сеанса на объект класса `HttpSession`. Сигнатура этого метода имеет следующую форму:

```
HttpSession getSession(String id)
```

`id` — идентификатор `(ID)` сеанса.

4.2 Обработка запросов и ответов HTTP

Класс `HttpServlet` обеспечивает специализированные методы, которые обрабатывают различные типы HTTP-запросов. Разработчик

обычно переопределяет один из этих методов. Это методы doDelete(), doGet(), doOptions(), doPost(), doPut() и doTrace(). При обработке ввода для форм обычно используются методы GET и POST.

4.2.1 Обработка GET-запросов HTTP

Ниже приведен пример сервлета, который обрабатывает GET-запрос HTTP. Сервлет вызывается из формы на Web-странице.

Пример содержит два файла: ColorGet.htm — определяет Web-страницу, и ColorGetServlet.java — определяет сервлет. Файл ColorGet.htm содержит форму, которая включает элемент выбора и кнопку. Стоит обратить внимание, что action-параметр тега формы (<form>) указывает на URL, который идентифицирует сервлет для обработки GET-запроса HTTP.

```
// файл ColorGet.htm
<html>
<body>
<center>
   <form name="Forml" action=
http://localhost:8080/servlet/ColorGetServIet">
<B>Цвет:</B>
 <select name="color" size="l">
  <option value="Red">Красный</option>
  <option value="Green">Зеленый</option>
  <option value="Blue">Синий</option>
 </select> <br><br>
  <input type=submit value="Отправить">
   </form>
</body>
</html>
```

Ниже показан исходный код для ColorGetServlet.java. Метод doGet() переопределен таким образом, чтобы обрабатывать любые GET-запросы HTTP, которые посылаются этому сервлету. Чтобы определить, что выбрал пользователь, используется метод getParameter() из HttpServletRequest. Затем формируется ответ.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ColorGetServlet
          extends HttpServlet {
public void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
String color = request.getParameter("color");
response.setContentType("text/html");
PrintWriter pw = response.getWriter();
pw.println("<B>The selected color is: ");
pw.println(color); pw.close () ;
    }
}
```

4.2.2 Обработка POST-запросов HTTP

Ниже приведен пример сервлета, который обрабатывает POST-запрос HTTP. Сервлет вызывается из формы на Web-странице. Пример содержит два файла: `ColorPost.htm` — определяет Web-страницу, и `ColorPostServlet.java` — определяет класс сервлета.

Исходный текст HTML (`ColorPost.htm`) показан ниже. Он идентичен `ColorGet.htm` за исключением того, что параметр `method` для тега `<form>` явно определяет использование POST-метода, а параметр `action` для того же тега определяет другой сервлет.

```
//файл ColorPost.htm
<html>
<body>
<center>
<form name="Form1" method="post"
ation="http://localhost:8080/servlet/
       ColorPostServlet"> <B>Color:</B>
  <select name="color" size="l">
    <option value="Red">Red</option>
   <option value="Green">Green</option>
```

```
   <option value="Blue">Blue</option>
  </select> <br><br>
   <input type=submit value="Submit">
</form>
</body>
</html>
```

Ниже показан исходный код файла `ColorPostServlet.java`. Метод `doPost()` переопределен для обработки любых POST-запросов HTTP, которые посылаются этому сервлету. Чтобы получить выбор, который был сделан пользователем, используется метод `getParameter()` из класса `HttpServletRequest`. Затем формируется ответ.

```
import java.io.*;
import зavax.serviet.*;
import javax.serviet.http.*;
public class ColorPostServlet
        extends httpServlet {
public void doPost(HttpServletRequest request,
            HttpServletResponse response,
 throws ServletExcepticn, IOException {
String color = request.getParameter("color");
response.setContentType("text/html");
PrintWriter pw = response.getWriter();
pw.println("<B>The selected color is: ");
pw.println(color); pw.close () ;
    }
}
```

Откомпилируйте сервлет и затем проверьте его работу, выполняя те же шаги, что описаны в предыдущем разделе.

Обратите внимание, что параметры для POST-запроса HTTP не включены как часть URL-адреса, который посылается Web-серверу. В рассмотренном примере URL, посылаемый из браузера на сервер, выглядит так:

```
http://localhost:8080/servlet/ColorPostServlet.
```

4.2.3 Отслеживание параметров сеанса

Гипертекстовый протокол передачи данных HTTP — протокол "без состояния" (stateless). Это означает, что каждый запрос независим от предыдущего. Однако в некоторых приложениях необходимо сохранить информацию о состоянии таким образом, чтобы можно было собрать информацию от нескольких взаимодействий между браузером и сервером. Такой механизм обеспечивают сеансы (sessions).

Сеанс может быть создан с помощью метода getSession() класса HttpServletRequest, который возвращает объект HttpSession. Данный объект может хранить набор связей, которые ассоциируют имена с объектами. Этими связями управляют методы putValue(), getValue(), getValueNames() и removeValue() класса HttpSession. Состояние сеанса поддерживается всеми сервлетами, которые связаны со специфическим клиентом.

Приведенный ниже исходный код сервлета иллюстрирует использование состояния сеанса. Метод getSession получает текущий сеанс. Новый сеанс создается, если он еще не существует. Чтобы получить объект, который связан с именем "date", вызывается метод getValue(). После этого будет возвращен объект Date, который инкапсулирует дату и время последнего обращения к данной странице. Затем создается объект Date, инкапсулирующий текущую дату и время. Наконец, вызывается метод putValue(), чтобы связать имя "date" с этим объектом.

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class DateServlet extends HttpServlet {
public void doGet(HttpServietRequest request,
HttpServletResponse response)
throws ServletException, IOException {
// получить HttpSession-объект
```

```
    HttpSession hs = request.getSession(true);
    // получить объект PrintWriter
    response.setContentType("text/html");
    PrintWriter pw = response.getWriter();
    pw.print("<B>") ;
    // показать дату/время последнего доступа Date
date = (Date)hs.getValue("date");
    if(date != null) {
    pw.print("Последний доступ: " +
              date +  "<br>");
    // показать текущую дату/время
    date = new Date();
    hs.putValue("date", date);
    pw.println ("Current date: " + date);
```

Когда первый раз происходит запрос к сервлету, браузер отображает одну строку с текущей датой и временем. При последующих вызовах отображаются две строки. Первая строка показывает дату и время последнего обращения к сервлету. Вторая строка — текущую дату и время.

4.3 Серверные страницы сценариев

4.3.1 Java Server Pages — общие сведения

Серверные страницы Java (Java Server Pages — JSP) — технология, которая позволяет смешивать статический HTML с динамически сгенерированным HTML. Большинство Web-страниц являются статическими, либо с динамической частью, ограниченной несколькими небольшими позициями. JSP позволяет создавать эти две части отдельно, например:

```
    <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
    <HTML>
    <HEAD><TITLE>Добро пожаловать!</TITLE></HEAD>
    <BODY>
    <H1>ДОБРО ПОЖАЛОВАТЬ!!!!!</H1>
    <SMALL>Welcome,
```

```
    <!–Имя пользователя будет "New User" для
первого посетителя страницы -->
    <% out.println(Добро пожаловать!!!)); %>
    Для доступа к Вашим настройкам перейдите
    <a HREF="Account-Settings.html">сюда.
</a></SMALL> <p> </BODY></HTML>
```

В приведенном фрагменте кода фрагмент, включенный между тегами <% и %>, и есть код JSP.

4.3.2 Синтаксис JSP

JSP могут содержать теги HTML, выражения, скриптлеты (фрагменты кода Java) и собственные теги. Обзор синтаксических элементов представлен в таблице 4.13.

Таблица 4.13 — Синтаксические элементы JSP

Элемент	Синтаксис	Пояснение
Выражение	<%= expression %>	Выражение вычисляется и помещается в выходной поток.
Скриптлет	<% code %>	Код вставляется в метод Service()
Объявление	<%! code %>	Код, который помещается в тело класса
Директива page	<%@ page att="val" %>	Глобальные установки JSP
Директива include	<%@ include file="url" %>	Локальный файл, который включается в текст JSP
Комментарий	<%-- Текст --%>	Комментарий
Действие include	<jsp:include page="relative URL" flush="true"/>	Включает содержимое файла в результате выполнения запроса
Действие jsp:forward	<jsp:forward page="relative URL"/>	Перемещение на другую страницу

4.3.3 Выражения JSP

Выражения предназначены для непосредственного вычисления значений и помещения их в ответ клиенту.

Пример:

```
Сегодня: <%= new java.util.Date() %>
```

В примере значение даты, прочитанное с помощью объекта класса `java.util.Date`, выводится в браузер.

Можно использовать встроенные переменные, такие как:

– `request` или `HttpServletRequest` (параметры запроса клиента);

– `response` или `HttpServletResponse` (ответ клиенту);

– `session` или `HttpSession` (параметры сеанса работы);

– `out` или `PrintWriter` (буферизированная версия объекта `JspWriter`) — вывод в браузер.

4.3.4 Скриптлеты

Скриптлет — это фрагмент кода, написанный с использованием Java и включенный в JSP. Синтаксис:

```
<% Java Code %>
```

Например:

```
<% String qData = request.getQueryString();
out.println("Получено значение даты: "
             + qData);  %>
```

Внутри скриптлета можно смешивать код Java и теги HTML, например, так:

```
<% if (Math.random() < 0.5) { %>
Сегодня <B> хороший </B> день!
<% } else { %>
Сегодня <B> неважный </B> день!
<% } %>
```

4.3.5 Объявления (декларации)

Объявления JSP позволяют объявлять переменные и методы внутри страницы. Синтаксис:

```
<%! Java Code %>
```

Например:

```
<%! private int accessCount = 0; %>
Эта страница будет перегружена через:
<%= ++accessCount %>
```

В данном примере объявлена переменная `accessCount`, а затем она же использована для вывода.

4.3.6 Директивы JSP

Директива JSP влияет на общую структуру класса `servlet`. У нее следующий синтаксис:

```
<%@ directive attribute="value" %>
```

Можно объединять многочисленные настройки для одной директивы следующим образом:

```
<%@ directive attribute1="value1"
attribute2="value2"
...
attributeN="valueN" %>
```

Есть два основных типа директив. Директива `page` позволяет оперировать объектами подобно импортируемым классам, модифицируя по заказу пользователя суперкласс сервлета. Директива `include` позволяет включать файл в класс сервлета во время трансляции JSP в сервлет.

Директива `page` позволяет устанавливать такие общие настройки JSP, как:

```
import="package.class"
```

или

```
import="package.class1,...,package.classN".
```

Эта настройка позволит указать пакеты для импорта, например:

```
<%@ page import="java.util.*" %>.
```

Настройка `import` — единственная, которую можно указывать многократно.

Настройка MIME задает тип вывода. По умолчанию это `text/html` (вывод в виде текста в формате HTML). Синтаксис:

```
contentType="MIME-Type"
```

или

contentType="MIME-Type; charset=Character-Set".

Например, директива <%@ page contentType="text/plain" %> действует также, как и скриптлет <% response.setContentType("text/plain"); %>.

Настройка isThreadSafe="true|false" указывает возможность использования потоков. Значение true (по умолчанию) показывает нормальное выполнение сервлета, когда многочисленные запросы обрабатываются одним сервлетом, если синхронизировать эти запросы с соответствующими переменными. Значение false указывает, что сервлет должен использовать интерфейс SingleThreadModel, который упорядочивает запросы и рассылает их соответствующим копиям сервлета.

Настройка session="true|false" позволяет задать параметры сеанса. Значение true (по умолчанию) показывает предопределенную переменную session (типа HttpSession) которая существует на данный момент, а в противном случае создается новый сеанс. Значение false указывает, что сеанс использоваться не будет, и поэтому обращение к переменной session выдаст ошибку, когда JSP будет преобразовываться в сервлет.

Настройка buffer="size kb|none" указывает размер буфера (объем памяти) для объекта JspWriter out. Обычно определяется сервером, но должно быть не менее 8 килобайт.

Настройка autoflush="true|false" по умолчанию равна true, и показывает, что буфер будет очищен при полном его заполнении. Если параметр установить в false (используется редко), то при переполнении буфера будет выработано исключение. Параметр некорректно устанавливается в false, если используется настройка buffer="none".

Настройка extends="package.class" показывает суперкласс для генерируемого сервлета. Используется с предупреждением, что сервер уже использует пользовательский суперкласс.

Настройка info="message" определяет строку, которая передается через метод getServletInfo().

Настройка `isErrorPage="true|false"` показывает, где отображается сообщение об ошибке: на текущей странице (`true`) или на новой странице (`false`). По умолчанию равна `false`.

Настройка `language="java"` идентифицирует используемый язык.

Директива `include` позволяет включать различные файлы в состав `JSP` на этапе трансляции в сервлет. Синтаксис:

```
<%@ include file="relative url" %>
```

Параметр `url` задает относительную ссылку на включаемый файл. Включаемые файлы могут содержать текст JSP, HTML, сценарии, директивы и действия.

Так, к примеру, многие сайты содержат панель навигации. Для того, чтобы исключить разделение страницы на фрагменты, эту панель обычно размещают внизу или вверху страницы в таблице. Можно один раз создать описание панели навигации и подключать ее с помощью директивы `include`, например:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD
  HTML 4.0 Transitional//EN">
<HTML> <HEAD>
<TITLE>СЕРВЛЕТЫ</TITLE>
<LINK REL=STYLESHEET
      HREF="Site-Styles.css"
      TYPE="text/css">  </HEAD>
<BODY>
<%@ include file="/navbar.html" %>
<!-- Содержимое страницы ... -->
</BODY> </HTML>
```

Примечание: после того, как директива `include` вставит файлы во время трансляции страницы, если панель навигации изменится, то придется ретранслировать все страницы JSP, которые ссылаются на эту ссылку. Включенные файлы изменяются часто, поэтому можно использовать `jsp:include` действие — это действие включает файл во время запроса к странице JSP.

4.3.7 Предопределенные переменные JSP

Для упрощения кода JSP и скриптлетов можно использовать восемь предопределенных переменных: `request`, `response`, `out`, `session`, `application`, `config`, `pageContext` и `page`. Ниже представлено описание для каждой из переменных:

1. Переменная request — аналог объекта класса `HttpServletRequest`, позволяет читать входные параметры (методом `getParameter`), параметры переданные пользователем (`GET`, `POST`, `HEAD` и др.), и собственные HTTP записи (`cookies`, `Referer` и др.). Строго говоря, `request` есть подкласс класса `ServletRequest` или его модификации `HttpServletRequest` (при использовании протокола HTTP).

2. Переменная response — аналог объекта класса `HttpServletResponse`, которая ассоциируется с ответом для клиента. Отметим, что выходной поток (см. `out` ниже) буферизирован, что позволяет легально устанавливать статус кода HTTP и заголовка ответа, даже если это не разрешено в сервлетах после посылки ответа клиенту.

3. Переменная out. Эта переменная использует объект `PrintWriter` для формирования ответа клиенту. Переменная `out` также может использоваться в скриптлетах, в этом случае выражение JSP автоматически помещается в выходной поток.

4. Переменная session. Эта переменная использует объект `HttpSession`. Создание `session` (сеанса работы) происходит автоматически. Если какое-либо выражение использует атрибут `session` в директиве `page`, сеанс закрывается, при этом могут возникнуть ошибки при переводе JSP в сервлет.

5. Переменная application. Эта переменная использует объект `ServletContext` через методы `getServletConfig()` и `getContext()`.

6. Переменная config. Представляет объект `ServletConfig` на текущей странице.

7. Переменная pageContext. В JSP введен новый класс, вызывающий `PageContext`, инкапсулирующий серверные расширения объектов типа `JspWriter`.

8. Переменная page — это простой синоним для `this`, который создается при выполнении сценариев.

4.3.8 Действия JSP

Действия (actions) JSP используют синтаксис XML (eXtensible Markup Language — расширенный язык разметки) для управления сервлетами. Основные действия:

- `jsp:include` — подключение файла во время приема страницы;
- `jsp:forward` — перемещение на страницу вперед;
- `jsp:plugin` — генерация специфического для данного браузера кода с использованием тегов `OBJECT` или `EMBED`.

4.4 Пример JSP

Ниже приведен пример совместного использования скриптлетов и директив и результат его выполнения (рисунок 4.1).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
<HTML> <HEAD>
<TITLE>Using JavaServer Pages</TITLE>
<META NAME="author" CONTENT="Marty Hall --
hall@apl.jhu.edu">
<META NAME="keywords"
      CONTENT="JSP,JavaServer Pages,servlets">
<META NAME="description"
      CONTENT="A quick example of the four main
JSP tags.">
<LINK REL=STYLESHEET
      HREF="My-Style-Sheet.css"
      TYPE="text/css"> </HEAD>
<BODY BGCOLOR="#FDF5E6" TEXT="#000000"
LINK="#0000EE"
      VLINK="#551A8B" ALINK="#FF0000">
<CENTER>
<TABLE BORDER=5 BGCOLOR="#EF8429">
  <TR><TH CLASS="TITLE">
```

```
    Using JavaServer Pages</TABLE>
    </CENTER>
    <P>
    Some dynamic content created using various JSP
mechanisms:
    <UL>
      <LI><B>Expression.</B><BR>
          Your hostname: <%=
request.getRemoteHost() %>.
    <LI><B>Scriptlet.</B><BR>
          <% out.println("Attached GET data: " +
             request.getQueryString()); %>
    <LI><B>Declaration (plus expression).</B><BR>
          <%! private int accessCount = 0; %>
          Accesses to page since server reboot: <%=
++accessCount %>
    <LI><B>Directive (plus expression).</B><BR>
          <%@ page import = "java.util.*" %>
          Current date: <%= new Date() %>
    </UL>
    </BODY> </HTML>
```

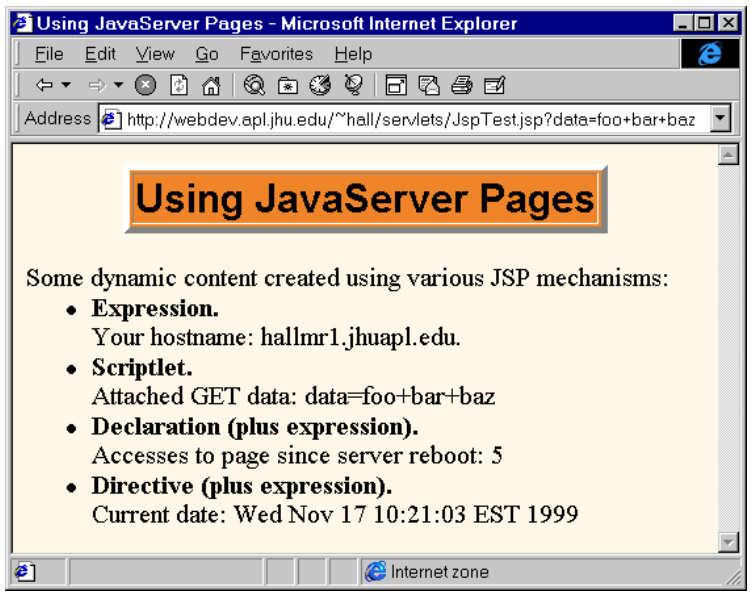


Рисунок 4.1 — Результат работы JSP

4.5 Задание к лабораторной работе

1. Создайте Web-страницу, на которой установите элементы ввода и две кнопки **Submit**. Форма должна вводить данные согласно описанию класса, разработанного в ЛР № 5 (II учебный семестр, ООП). Все поля должны иметь значения по умолчанию.

2. Форма на Web-странице должна передавать данные:

– в сервлет при нажатии на первую кнопку **Submit**;

– на JSP при нажатии на вторую кнопку **Submit**.

Полученные данные выводите клиенту в виде таблицы. Для хранения предыдущих данных используйте сеанс (session).

3. Распечатайте тексты:

– Web-страницы с формой;

– программный код сервлета;

– текст JSP;

– результаты выполнения сервлета из браузера;

– результаты выполнения JSP из браузера.

4.6 Контрольные вопросы

1. Что такое сервлет? Какова структура сервлета?

2. Жизненный цикл сервлета. Перечислите методы и порядок их выполнения.

3. Интерфейсы пакета javax.servlet – перечислите, укажите назначение.

4. Классы пакета javax.servlet – перечислите, укажите назначение.

5. Методы интерфейса Servlet. Обработка сервлетом GET-запросов.

6. Методы интерфейса ServletContext. Обработка сервлетом GET-запросов.

7. Методы интерфейса ServletRequest. Обработка сервлетом GET-запросов.

8. Методы интерфейса ServletResponse. Обработка сервлетом GET-запросов.

9. Как прочитать параметры сервлета? Приведите пример.

10. Интерфейсы пакета javax.servlet.http. Обработка сервлетом GET-запросов

11. Классы пакета javax.servlet.http. Обработка сервлетом GET-запросов

12. Как прочитать и настроить параметры сеанса?

13. Обработка сервлетом GET-запросов. Приведите пример.

14. Обработка сервлетом POST-запросов. Приведите пример.

15. Что такое JSP? Какие элементы может содержать JSP?

16. Выражения и декларации JSP. Что в них общего и чем они отличаются?

17. Скриптлеты — назначение, синтаксис, пример применения.

18. Директивы JSP. Перечислите, укажите назначение, приведите пример применения.

19. Предопределенные переменные JSP — перечислите, укажите назначение, приведите пример использования.

20. Действия JSP. Приведите примеры их использования.

5 ЛАБОРАТОРНАЯ РАБОТА №5

Тема работы: графика Java 2D.

Цель работы: научиться производить графические построения с использованием пакета Java 2D.

5.1 Обзор Java 2D API

Интерфейсы Java 2D API, представленные начиная с JDK 1.2, предусматривают для программистов Java широкие возможности работы с двумерными графическими фигурами, текстом и изображениями через расширения инструментария Abstract Windowing Toolkit (AWT). Эта библиотека отображений поддерживает формирование линий, текста и изображений в виде гибкой многофункциональной структуры для разработки богатого пользовательского интерфейса, графических программ и редакторов изображений.

Интерфейсы Java 2D API предусматривают:

– единообразную модель отображения (uniform rendering model) для экранов и принтеров;

– широкий спектр геометрических примитивов — кривых, прямоугольников, эллипсов, а также механизм для отображения любой геометрической фигуры;

– механизм для обнаружения вмешательства пользователя (hit detection) на поверхности фигур, текста или изображений;

– модель композиции (compositing model), которая предусматривает разные варианты отображения при перекрытии фигур;

– расширенную поддержку цвета (color support), позволяющую управлять цветопередачей;

– поддержку печати сложных документов.

5.1.1 Модель отображения Java 2D Rendering

Система рисования Java определяет, когда именно и как программы могут рисовать. Если компонент должен быть показан, его методы `paint()` или `update()` автоматически вызываются с соответствующим контекстом `Graphics`.

Класс `Graphics2D` расширяет класс `Graphics` и предоставляет доступ к расширенным возможностям работы с графикой и

отображениями интерфейса Java 2D API.

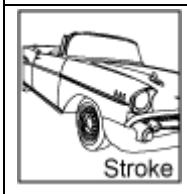
Чтобы использовать возможности Java 2D API, необходимо преобразовать объект `Graphics`, переданный методу отображения компонента, к объекту `Graphics2D`.

```
public void paint (Graphics g) {
    Graphics2D g2 = (Graphics2D) g;
    ...
}
```

5.1.2 Контекст отображения Graphics2D

Коллекцию атрибутов состояния, связанных с объектом `Graphics2D`, называют `Graphics2D rendering context` — контекстом отображения Graphics2D. Для отображения текста, фигур или картинок необходимо сначала задать контекст отображения, а затем вызвать один из методов отображения, например `draw` или `fill`. В таблице 5.1 показаны атрибуты, содержащиеся в контексте отображения `Graphics2D rendering context`.

Таблица 5.1 — Атрибуты, содержащиеся в контексте отображения Graphics2D

Иллюстрация	Атрибуты
	Стиль *pen style (перо)* относится к контуру изображения. Атрибут *stroke (контур)* позволяет рисовать линии любой толщины и применять шаблоны пунктира для прерывистой линии.
	Стиль *fill style (заливка)* относится к внутреннему содержимому фигуры. Атрибут *paint (рисовать)* позволяет заполнять контур сплошным цветом, заливкой с переходом цветов из одного в другой, или текстурой в соответствии с заданным шаблоном.
	Стиль *compositing style (композиция)* применяется при отображении объектов, построенных из нескольких перекрывающихся объектов.

Продолжение таблицы 5.1

Иллюстрация	Атрибуты
	Стиль *transform (трансформация)* действует во время отображения и преобразует объект из пользовательского пространства в систему координат отображающего устройства (device-space coordinates). С помощью этого атрибута можно по выбору задать преобразование переноса (translation), поворота (rotation), масштабирования (scaling) или сжатия (shearing).
	Стиль *clip (клип)*, ограничивающий отображение некоторой областью внутри фигуры `Shape`, служит для определения вырезанной области клипа – clipping path. Для определения клипа может быть использован любой объект `Shape`.
	Указание *font (шрифт)* служит для преобразования текстовых строк в изображения.
	*Rendering hints ("советы")* задают правила предпочтения при отображении в условиях выбора между качеством и быстродействием. Например, Вы можете указать, нужно ли использовать спектральное сглаживание или подавление помех.

Для установки атрибутов контекста отображения `Graphics2D` (см. табл. 5.1) используют следующие методы:

– `setStroke();`
– `setPaint();`
– `setComposite();`
– `setTransform();`
– `setClip();`
– `setFont();`
– `setRenderingHints()`.

Для установки атрибута следует использовать объект соответствующего класса. Например, чтобы изменить атрибут заливки

на градиентное заполнение "синий-зеленый", необходимо создать объект GradientPaint и затем вызвать метод setPaint().

```
gp = new GradientPaint
          (0f,0f,blue,0f,30f,green);
g2.setPaint(gp);
```

Объект класса Graphics2D содержит ссылки на объекты своих атрибутов. Если необходимо изменить объект какого-либо атрибута, являющегося частью контекста Graphics2D context, нужно вызвать соответствующий метод setXXX для модификации контекста. Модификация объекта атрибута непосредственно во время отображения может привести к непредвиденным результатам.

5.1.3 Методы отображения

Класс Graphics2D предусматривает следующие общие методы отображения, которые можно использовать для рисования геометрических фигур, текста или изображений:

– draw() — отображает контур любого геометрического примитива, используя атрибуты stroke и paint;

– fill() — отображает любой геометрический примитив, заполняя его содержимое цветовой заливкой или шаблоном, задаваемым с помощью атрибута paint;

– drawString() — отображает любую текстовую строку. Атрибут font используется для преобразования строк в изображения, которые затем заполняются цветом или шаблоном, задаваемым с помощью атрибута paint;

– drawImage() — отображает указанное изображение (image).

Кроме того, класс Graphics2D поддерживает методы отображения класса Graphics для конкретных фигур — например, drawOval() или fillRect().

5.2 Системы координат

Пакет Java 2D поддерживает две системы координат:

– User space (пространство пользователя) — пространство, в котором задаются графические примитивы;

– Device space (пространство устройства) — система координат

устройства вывода: экрана, окна или принтера.

Пространство пользователя представляет собой независимую от устройств логическую систему координат, которую использует программа. Все геометрические фигуры, передаваемые в отображающие процедуры Java 2D, задаются в системе координат пользователя.

При использовании стандартной передачи изображения из пространства пользователя в пространство устройства начало отсчета пространства пользователя находится в верхнем левом углу области изображения компонента. Координата *x* возрастает вправо, а координата *y* возрастает вниз (рис. 5.1).

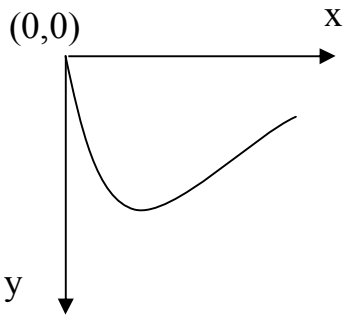


Рисунок 5.1 — Система координат

Пространство устройства представляет собой независимую от конкретного устройства систему координат, изменяющуюся в зависимости от используемого устройства отображения. Хотя системы координат для окна или экрана могут отличаться от системы координат для принтера, эти различия остаются невидимыми для Java-программ. Необходимые преобразования между пользовательским пространством и пространством устройства выполняются автоматически во время отображения.

5.3 Формирование и заполнение графических примитивов

Классы графических примитивов: точки, линии, кривые, дуги, прямоугольники и эллипсы определены в библиотеке `java.awt.geom` (табл. 5.2).

Таблица 5.2 — Классы в библиотеке java.awt.geom

Класс	Описание
`RectangularShape`	Абстрактный класс для фигур — не может быть использован напрямую
`Arc2D`	Содержит набор методов для построения дуг
`Ellipse2D`	Содержит набор методов для построения эллипсов и окружностей
`Rectangle2D`	Содержит набор методов для построения прямоугольников
`RoundRectangle2D`	Содержит набор методов для построения прямоугольников со скругленными краями
`Line2D`	Содержит набор методов для построения линий
`Dimension2D`	Содержит набор методов для установки размеров фигур
`Point2D`	Содержит набор методов для построения точек
`QuadCurve2D`	Содержит набор методов для построения кривых второго порядка
`CubicCurve2D`	Содержит набор методов для построения кривых третьего порядка
`Area`	Позволяет выполнять над двумя объектами `Shape` логические операции
`GeneralPath`	Позволяет составить комбинированную фигуру, задавая серию позиций вдоль границ фигуры

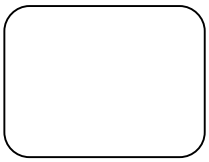
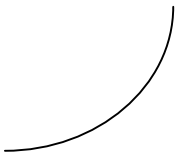
Исключая `Point2D` и `Dimension2D`, каждый из классов графических примитивов реализует интерфейс `Shape`, который предусматривает общий набор методов для описания и проверки двумерных геометрических объектов.

С помощью этих классов можно создать нужную геометрическую фигуру и отобразить ее через `Graphics2D` вызовом метода `draw()` или `fill()`.

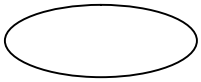
Графические примитивы `Rectangle2D`, `RoundRectangle2D`, `Arc2D` и `Ellipse2D` являются наследниками класса `RectangularShape`, определяющего методы для объектов всех объектов `Shape`, которые могут быть описаны в границах прямоугольника.

Внешний вид основных геометрических фигур и их конструкторы соответствующих классов приведены в таблице 5.3.

Таблица 5.3 — Конструкторы для создания основных геометрических фигур

Фигура	Конструкторы
	Line2D.Double(double X1,double Y1, double X2, double Y2) где X1,Y1 – координаты начальной точки, X2,Y2 – конечной точки прямой. Line2D.Double (Point2D p1, Point2D p2) где p1, p2 – начальная и конечная точки соответственно, определенные как объекты класса Point2D.
	Rectangle2D.Double(double x, double y, double w, double h) где x, y – координаты верхнего левого угла прямоугольника; w и h – его ширина и высота соответственно.
	RoundRectangle2D.Double (double x, double y, double w, double h, double arcw, double arch) где x, y – координаты верхнего левого угла; w, h – ширина и высота фигуры; arcw, arch – ширина и высота дуги соответственно.
	Arc2D.Double(double x, double y, double w, double h, double start, double extent, int type) где x, y – координаты верхнего левого угла; w, h – ширина и высота полного эллипса (не рассматривая угловые протяженности); start – стартовый угол дуги в градусах; extent - угловая протяженность дуги в градусах; type - тип закрытия дуги (CHORD, OPEN, PIE). Arc2D.Double(Rectangle2D ellipseBounds, double start, double extent, int type) где ellipseBounds – прямоугольник, в который будет вписана дуга; start – стартовый угол дуги в градусах; extent - угловая протяженность дуги в градусах; type - тип закрытия дуги.

Продолжение таблицы 5.3

Фигура	Конструкторы
	Ellipse2D.Double(double x, double y, double w, double h) где x, y – координаты верхнего левого угла; w, h – ширина и высота полного эллипса.

Ниже приведен пример, демонстрирующий, как можно отобразить основные геометрические фигуры, используя методы `Graphics2D draw()` (строки 67, 72, 77, 82, 87) и `fill()` (строки 92, 103, 109, 116).

Данный пример использует реализации двойной точности (double-precision implementations) для геометрических классов. Результат работы программы приведен на рисунке 5.2.

```
1    import java.awt.*;
2    import java.awt.event.*;
3    import java.awt.geom.*;
4    import javax.swing.*;
5    public class ShapesDemo2D extends JApplet {
6    final static int maxCharHeight = 15;
7    final static int minFontSize = 6;
8    final static Color bg = Color.white;
9    final static Color fg = Color.black;
10   final static Color red = Color.red;
11   final static Color white = Color.white;
12   final static BasicStroke stroke = new
     BasicStroke(2.0f);
13   final static BasicStroke wideStroke = new
     BasicStroke(8.0f);
14   final static float dash1[] = {10.0f};
15   final static BasicStroke dashed = new
     BasicStroke(1.0f,        BasicStroke.CAP_BUTT,
     BasicStroke.JOIN_MITER,10.0f, dash1, 0.0f);
16   Dimension totalSize;
17   FontMetrics fontMetrics;
18   public void init() {
19   //задаем цвет фона и надписи
```

```
20   setBackground(bg);
21   setForeground(fg);
22   }
23   FontMetrics  pickFont(Graphics2D  g2,  String
     longString,int xSpace) {
24   boolean fontFits = false;
25   Font font = g2.getFont();
26   FontMetrics           fontMetrics           =
     g2.getFontMetrics();
27   int size = font.getSize();
28   String name = font.getName();
29   int style = font.getStyle();
30   while ( !fontFits ) {
31   if      (      (fontMetrics.getHeight()      <=
     maxCharHeight)
32   &&   (fontMetrics.stringWidth(longString)   <=
     xSpace) ) {
33   fontFits = true;
34   }
35   else {
36   if ( size <= minFontSize ) {
37   fontFits = true;
38   }
39   else {
40   g2.setFont(font = new Font(name,
41   style,
42   --size));
43   fontMetrics = g2.getFontMetrics();
44   }
45   }
46   }
47   return fontMetrics;
48   }
49   public void paint(Graphics g) {
50   Graphics2D g2 = (Graphics2D) g;
```

```
51   g2.setRenderingHint
     (RenderingHints.KEY_ANTIALIASING,
     RenderingHints.VALUE_ANTIALIAS_ON);
52   Dimension d = getSize();
53   int gridWidth = d.width / 6;
54   int gridHeight = d.height / 2;
55   fontMetrics     =     pickFont(g2,     "Filled
     RoundRectangle2D", gridWidth);
56   Color fg3D = Color.lightGray;
57   g2.setPaint(fg3D);
58   g2.draw3DRect(0,  0,  d.width  -  1,  d.height  -
     1, true);
59   g2.draw3DRect(3,  3,  d.width  -  7,  d.height  -
     7, false);
60   g2.setPaint(fg);
61   int x = 5;
62   int y = 7;
63   int rectWidth = gridWidth - 2*x;
64   int    stringY    =    gridHeight    -    3    -
     fontMetrics.getDescent();
65   int      rectHeight      =      stringY      -
     fontMetrics.getMaxAscent() - y - 2;
66   // рисуем линию
67   g2.draw(new  Line2D.Double(x,  y+rectHeight-1,
     x + rectWidth, y));
68   g2.drawString("Line2D", x, stringY);
69   x += gridWidth;
70   // рисуем прямоугольник
71   g2.setStroke(stroke);
72   g2.draw(new      Rectangle2D.Double(x,      y,
     rectWidth, rectHeight));
73   g2.drawString("Rectangle2D", x, stringY);
74   x += gridWidth;
75   //   рисуем   прямоугольник   со   скругленными
     краями
```

```
76   g2.setStroke(dashed);
77   g2.draw(new   RoundRectangle2D.Double(x,   y,
     rectWidth, rectHeight, 10, 10));
78   g2.drawString("RoundRectangle2D",          x,
     stringY);
79   x += gridWidth;
80   // рисуем дугу
81   g2.setStroke(wideStroke);
82   g2.draw(new   Arc2D.Double(x,   y,   rectWidth,
     rectHeight, 90, 135, Arc2D.OPEN));
83   g2.drawString("Arc2D", x, stringY);
84   x += gridWidth;
85   // рисуем элипс
86   g2.setStroke(stroke);
87   g2.draw(new Ellipse2D.Double(x, y, rectWidth,
     rectHeight));
88   g2.drawString("Ellipse2D", x, stringY);
89   x += gridWidth;
90   // рисуем закрашенный прямоугольник
91   g2.setPaint(red);
92   g2.fill(new      Rectangle2D.Double(x,      y,
     rectWidth, rectHeight));
93   g2.setPaint(fg);
94   g2.drawString("Filled    Rectangle2D",     x,
     stringY);
95   x += gridWidth;
96   // задаем координаты новой строки
97   x = 5;
98   y += gridHeight;
99   stringY += gridHeight;
100  //рисуем     закрашенный     прямоуг-к     со
     скругл.краями
101  GradientPaint      redtowhite      =      new
     GradientPaint(x,y,red,x+rectWidth, y,white);
102  g2.setPaint(redtowhite);
```

```
103  g2.fill(new   RoundRectangle2D.Double(x,   y,
     rectWidth,rectHeight, 10, 10));
104  g2.setPaint(fg);
105  g2.drawString("Filled  RoundRectangle2D",  x,
     stringY);
106  x += gridWidth;
107  // рисуем закрашенную дугу
108  g2.setPaint(red);
109  g2.fill(new  Arc2D.Double(x,  y,  rectWidth,
     rectHeight, 90, 135, Arc2D.OPEN));
110  g2.setPaint(fg);
111  g2.drawString("Filled Arc2D", x, stringY);
112  x += gridWidth;
113  // рисуем закрашенный эллипс
114  redtowhite                =                new
     GradientPaint(x,y,red,x+rectWidth, y,white);
115  g2.setPaint(redtowhite);
116  g2.fill    (new    Ellipse2D.Double(x,    y,
     rectWidth, rectHeight));
117  g2.setPaint(fg);
118  g2.drawString("Filled     Ellipse2D",     x,
     stringY);
119  x += gridWidth;
120  }
121  public static void main(String s[]) {
122  JFrame f = new JFrame("Shapes2D");
123  f.addWindowListener(new WindowAdapter() {
124  public void windowClosing(WindowEvent e) {
125  System.exit(0);}
126  });
127  JApplet applet = new ShapesDemo2D();
128  f.getContentPane().add("Center", applet);
129  applet.init();
130  f.pack();
131  f.setSize(new Dimension(550,300));
```

```
132   f.show();
133   }
134   }
```

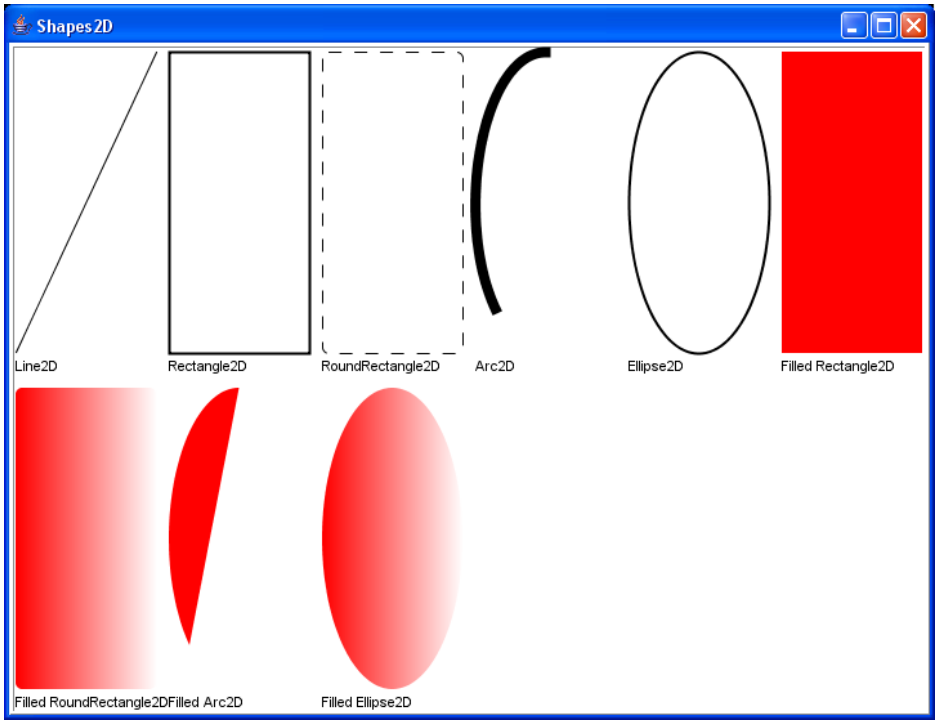


Рисунок 5.2 — Вывод основных фигур методами draw() и fill()

Для рисования «объемного» прямоугольника используется метод `draw3DRect()` (строки 58–59). Его использование отличается от обычного прямоугольника логическим параметром `raised`, отвечающим за наличие объема.

Изменяя атрибуты формы и отображения в контексте `Graphics2D`, можно задать графическим примитивам особые стили, формы и содержание. Например, можно задать стили линий для фигур с помощью соответствующего объекта `Stroke` (строки 12, 13, 15) и вызовом `setStroke()` (строки 71, 76, 81, 86) добавить его к контексту `Graphics2D` перед отображением. Аналогично, можно применить градиентное заполнение к `Shape` созданием объекта `GradientPaint` (строка 101) и добавлением его к контексту `Graphics2D` путем вызова `setPaint()` (строки 102, 115) перед тем, как отобразите `Shape`. Более подробно о стилях линий и способах заполнения сказано ниже.

Переменные rectWidth и rectHeight в данном примере определяют размеры пространства, где рисуется каждая фигура (в пикселях) (строка 63, 65). Переменные x и y изменяются для каждой фигуры так, чтобы фигуры вписывались в формат сетки размером 6x2 (строки 53-54).

Класс GeneralPath позволяет создать комбинированную фигуру, задавая серию позиций вдоль границ фигуры, которые могут быть соединены сегментами прямых или кривых линий.

Вначале создается пустой объект класса GeneralPath конструктором по умолчанию GeneralPath() или объект, содержащий одну фигуру, конструктором GeneralPath (Shape sh).

К этому объекту можно добавлять фигуры методом append(Shape sh, boolean connect). Если параметр connect равен true, то новая фигура соединяется с предыдущими фигурами с помощью текущего пера.

В объекте всегда есть текущая точка. По умолчанию ее координаты (0, 0). Для их изменения необходимо использовать метод moveTo(float x, float y). От текущей точки к точке (x, y) можно провести отрезок прямой методом lineTo(float x, float y). Текущей точкой после этого становится точка (x, y). Начальную и конечную точки можно соединить методом closePath().

5.4 Стили линии

Стили линий определяются атрибутом stroke, для установки которого нужно создать объект BasicStroke и передать его в метод Graphics2D setStroke().

Основной конструктор для стилей имеет следующий синтаксис: BasicStroke(float width, int cap, int join, float miter, float[] dash, float dashBegin),

где width — толщина пера в пикселях;

cap — оформление конца линии (одна из констант: CAP_ROUND — закругленный конец линии; CAP_SQUARE — квадратный конец линии; CAP_BUTT — оформление отсутствует);

join — способ сопряжения линий (одна из констант: JOIN_ROUND – линии сопрягаются дугой окружности; JOIN_BEVEL — линии сопрягаются отрезком прямой, перпендикулярным биссектрисе угла между линиями; JOIN_MITER — линии просто стыкуются);

miter — расстояние между линиями, начиная с которого применяется сопряжение JOIN_MITER;

dash[] — массив длин штрихов и промежутков между штрихами (элементы массива с четными индексами задают длину штриха в пикселях, элементы с нечетными индексами — длину промежутка);

dashBegin — индекс, начиная с которого повторяются элементы массива dash.

Остальные конструкторы задают некоторые характеристики по умолчанию:

- BasicStroke(float width, int cap, int join, float miter) — сплошная линия;

- BasicStroke(float width, int cap, int join) — сплошная линия с сопряжением JOIN_ROUND или JOIN_BEVEL; для сопряжения JOIN_MITER задается значение miter = 10.0f;

- BasicStroke(float width) — прямой обрез CAP_SQUARE и сопряжение JOIN_MITER со значением miter = 10.0f;

- BasicStroke() — ширина 1. 0f.

Пример программы, использующей различные стили линий, приведен ниже:

```
1   import java.awt.*;
2   import java.awt.geom. *;
3   import java.awt.event.*;
4   class StrokeTest extends Frame{
5   StrokeTest(String s) {
6   super (s) ;
7   setSize(500, 400);
8   setVisible(true);
9   addWindowListener(new WindowAdapter(){
10  public void windowClosing(WindowEvent ev){
```

```
11  System.exit(0);
12  }
13  });
14  }
15  public void paint(Graphics gr){
16  Graphics2D g = (Graphics2D)gr;
17  g.setFont(new Font("Serif", Font.PLAIN, 15));
18  BasicStroke   penl   =   new   BasicStroke(20,
    BasicStroke.CAP_BUTT,
    BasicStroke.JOIN_MITER,30);
19  BasicStroke   pen2   =   new   BasicStroke(20,
    BasicStroke.CAP_ROUND,
    BasicStroke.JOIN_ROUND);
20  BasicStroke   pen3   =   new   BasicStroke(20,
    BasicStroke.CAP_SQUARE,
    BasicStroke.JOIN_BEVEL);
21  float[] dashl = {5, 20};
22  BasicStroke   pen4   =   new   BasicStroke(10,
    BasicStroke.CAP_ROUND, BasicStroke.JOIN_BEVEL,
    10, dashl, 0);
23  float[] dash2 = {10, 5, 5, 5};
24  BasicStroke   pen5   =   new   BasicStroke(10,
    BasicStroke.CAP_BUTT,  BasicStroke.JOIN_BEVEL,
    10, dash2, 0);
25  g.setStroke(penl);
26  g.draw(new   Rectangle2D.Double(50,   50,   50,
    50));
27  g.draw(new Line2D.Double(50, 180, 150, 180));
28  g.setStroke(pen2);
29  g.draw(new   Rectangle2D.Double(200,   50,   50,
    50));
30  g.draw(new Line2D.Double(50, 230, 150, 230));
31  g.setStroke(pen3);
32  g.draw(new   Rectangle2D.Double(350,   50,   50,
    50));
```

```
33  g.draw(new Line2D.Double(50, 280, 150, 280));
34  g.drawString("JOIN_MITER", 40, 130);
35  g.drawString("JOIN_ROUND", 180, 130);
36  g.drawString("JOIN_BEVEL", 330, 130);
37  g.drawString("CAP_BUTT", 170, 190);
38  g.drawString("CAP_ROUND", 170, 240);
39  g.drawString("CAP_SQUARE", 170, 290);
40  g.setStroke(pen5);
41  g.draw (new Line2D.Double(50, 330, 250, 330));
42  g.setStroke(pen4);
43  g.draw(new Line2D.Double(50, 360, 250, 360));
44  g.drawString("{10, 5, 5, 5,...}", 260, 335);
45  g.drawString("(5, 20,...}", 260, 365);
46  }
47  public static void main(String[] args){
48  new StrokeTest("Моя программа");
49  }
50  }
```

В данном примере определено пять объектов `BasicStroke` с разными характеристиками (строки 18-20, 22, 24). Для создания стилей `pen4` и `pen5` сформированы массивы `dash1` и `dash2` (строки 21, 23), определяющие длину штрихов и промежутков между ними. Для применения стиля необходимо добавить объект `Stroke` к контексту `Graphics2D` методом `setStroke()` (строки 25, 28, 31, 40, 42). Затем отображаются фигуры (строки 26, 27, 29, 30, 32, 33, 41, 43). Подписи фигур с настройками перьев выводятся с помощью метода `drawstring()` (строки 34-39, 44-45).

Результат работы программы приведен на рисунке 5.3.

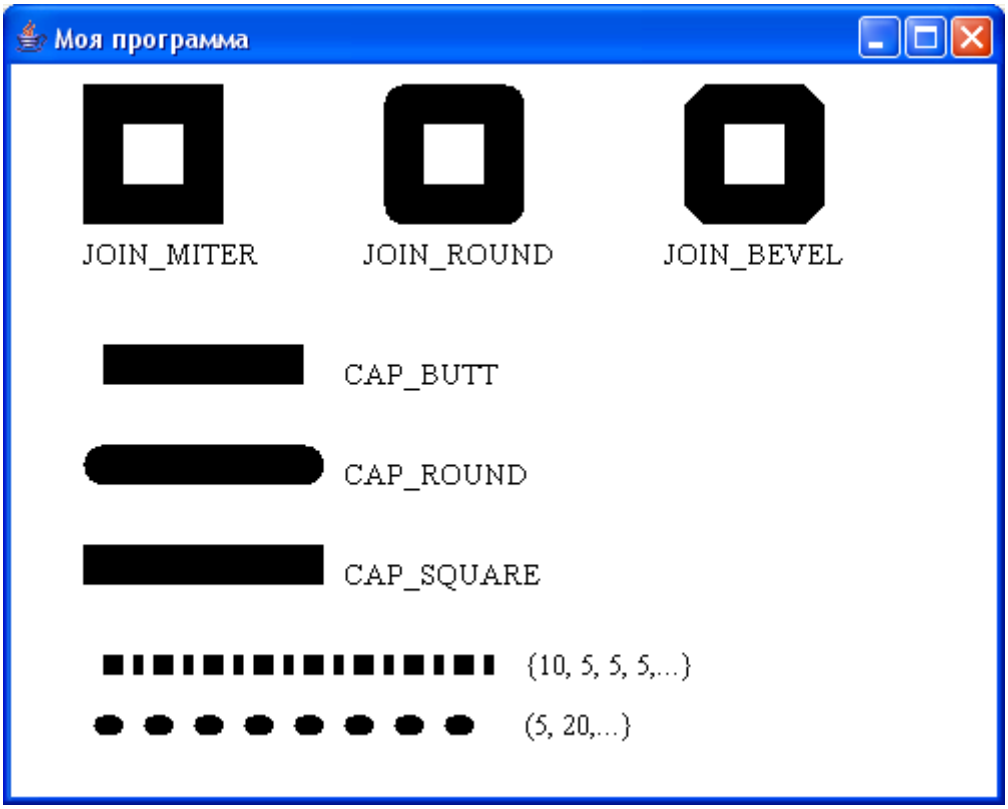


Рисунок 5.3 — Перья с различными характеристиками

5.5 Шаблоны заполнения

Шаблоны заполнения определены атрибутом `paint` в контексте отображения `Graphics2D`. Для установки атрибута `paint` необходимо создать экземпляр объекта, который реализует (implements) интерфейс `Paint` и передает его в метод `Graphics2D setPaint()`.

Интерфейс `Paint` реализует три класса: `Color`, `GradientPaint` и `TexturePaint`. Классы `GradientPaint` и `TexturePaint` есть в версиях JDK 1.2 и старше.

Некоторые конструкторы класса `Color`:

– `Color(int r, int g, int b)` — создает непрозрачный sRGB цвет с определенными красными, зелеными, и синими интенсивностями в диапазоне от 0 до 255.

– `Color(int r, int g, int b, int a)` — создает прозрачный RGB цвет с определенной красной, зеленой, синей величиной и величиной прозрачности α в области от 0 до 255.

Конструкторы класса `GradientPaint` представлены ниже.

`GradientPaint(float x1, float y1, Color c1,`

float x2, float y2, Color c2)

Для получения градиентной заливки в двух точках *M* и *N* устанавливаются разные цвета. В точке *M*(*x1, y1*) задается цвет c1, в точке *N*(*x2, y2*) — цвет c2. Цвет заливки изменяется плавно от c1 к c2 вдоль прямой, соединяющей точки *M* и *N*, оставаясь постоянным вдоль каждой прямой, перпендикулярной прямой *MN*. При этом вне отрезка *MN* цвет остается постоянным: за точкой *M* — цвет c1, за точкой *N* — цвет c2.

GradientPaint(float x1, float y1, Color c1, float x2, float y2, Color c2, boolean cyclic)

Второй конструктор при задании параметра cyclic == true повторяет заливку полосы *MN* во всей заливаемой фигуре.

Другие два конструктора задают точки как объекты класса Point2D:

GradientPaint(Point2D pt1, Color c1, Point2D pt2, Color c2)

GradientPaint(Point2D pt1, Color c1, Point2D pt2, Color c2, boolean cyclic)

Класс TexturePaint работает сложнее. Сначала создается буфер — объект класса BufferedImage из пакета java.awt.image. При этом необходимо получить его графический контекст, управляемый экземпляром класса Graphics2D. Этот экземпляр можно получить методом createGraphics() класса BufferedImage. Графический контекст буфера заполняется фигурой, которая будет служить образцом заполнения.

Затем из буфера создается объект класса TexturePaint. При этом задают размеры образца заполнения. Конструктор выглядит так:

TexturePaint(BufferedImage bi, Rectangle2D anchor)

где bi – объект класса BufferedImage;

anchor – прямоугольник, размеры которого соответствуют размерам образца заполнения.

После создания заливки — объекта класса Color, GradientPaint или TexturePaint — она устанавливается в

графическом контексте методом `setPaint(Paint p)` и используется в дальнейшем методом `fill(Shape sh)`.

Ниже приведен пример, в котором прямоугольник закрашивается текстурной заливкой, а окружность — градиентной:

```
1   import java.awt.*;
2   import java.awt.geom.*;
3   import java.awt.image.*;
4   import java.awt.event.*;
5   class PaintTest extends Frame{PaintTest(String
    s){ super(s);
6   setSize(310, 310);
7   setVisible(true);
8   addWindowListener(new WindowAdapter(){
9   public void windowClosing(WindowEvent ev){
10  System.exit(0);
11  }
12  });
13  }
14  public void paint(Graphics gr){
15  Graphics2D g = (Graphics2D)gr;
16  BufferedImage bi = new BufferedImage(20,20,
    BufferedImage.TYPE_INT_RGB );
17  Graphics2D big = bi.createGraphics();
18  big.draw(new Line2D.Double(0.0, 0.0, 10.0,
    10.0));
19  big.draw(new Line2D.Double(0.0, 10.0, 10.0,
    0.0));
20  TexturePaint tp = new TexturePaint(bi,new
    Rectangle2D.Double(0.0, 0.0, 10.0, 10.0));
21  g.setPaint(tp);
22  g.fill(new Rectangle2D.Double(50, 50, 200,
    200));
23  GradientPaint gp =new GradientPaint(100, 100,
    Color.white,150, 150, Color.black, true);
24  g.setPaint(gp);
```

```
25  g.fill(new  Ellipse2D.Double(100,  100,  200,
    200));
26  }
27  public static void main(String[] args){
28  new PaintTest(" Способы заливки");
29  }
30  }
```

Результат программы приведен на рисунке 5.4.

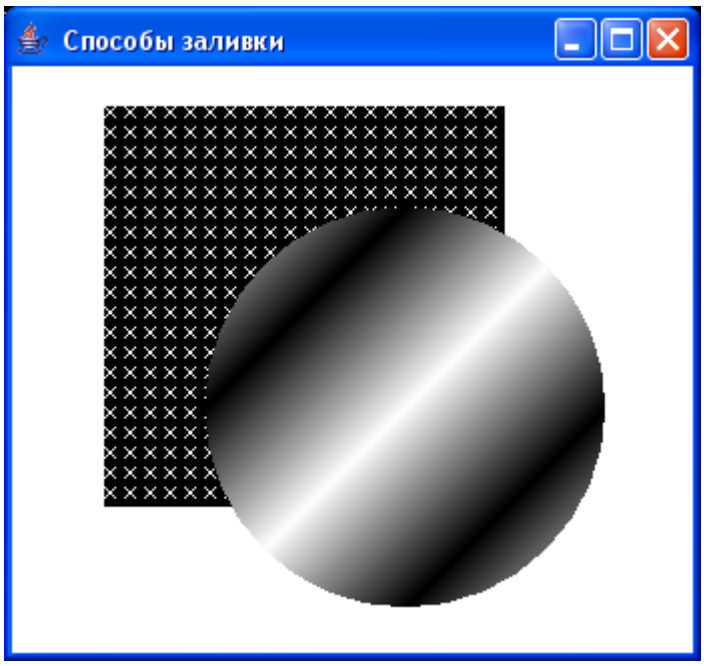


Рисунок 5.4 — Способы заливки

Для создания текстурной заливки создается объект `bi` класса `BufferedImage` (строка 16), затем получается его графический контекст `big` (строка 17). В нем рисуется фигура — шаблон заполнения (строки 18-19). Далее создается объект `tp` класса `TexturePaint`, которому передаются шаблон заполнения `bi` и размер образца заполнения (строка 20). В строке 23 создается градиентная заливка — объект класса `GradientPaint`. Заливки устанавливаются в графическом контексте методом `setPaint()` (строки 21, 24) и используются в дальнейшем методом `fill()` (строки 22, 25).

5.6 Вывод текста средствами Java 2D

Если необходимо вывести в графическом режиме строку текста, используют метод `drawString()`, а для задания шрифта в строке — метод `setFont()`.

При использовании Java 2D API шрифты задаются с использованием экземпляра класса `Font`. Можно определить доступные в системе шрифты вызовом статического метода `GraphicsEnvironment.getLocalGraphicsEnvironment()` и затем запросив возвращенное окружение `GraphicsEnvironment`.

Метод `getAllFonts()` возвращает массив, который содержит экземпляры `Font` для всех доступных в системе шрифтов, а метод `getAvailableFontFamilyNames()` возвращает список доступных семейств шрифтов.

Шрифт (объект класса `Font`) кроме имени, стиля и размера имеет еще полтора десятка атрибутов: подчеркивание, перечеркивание, наклон, цвет шрифта и цвет фона, ширину и толщину символов, аффинное преобразование, расположение слева направо или справа налево.

Атрибуты шрифта задаются как статические константы класса `TextAttribute`. Наиболее часто используемые атрибуты перечислены в таблице 5.4.

Таблица 5.4 — Атрибуты шрифта

Атрибут	Значение
BACKGROUND	Задает цвет фона.
FOREGROUND	Задает цвет текста.
BIDI_EMBEDDED	Задает уровень вложенности просмотра текста. Целое от 1 до 15.
CHAR_REPLACEMENT	Задает вспомогательную фигуру для отображения в тексте вместо символа.
FAMILY	Семейство шрифта.
FONT	Создает шрифт - объект класса Font
JUSTIFICATION	Задает допуск при выравнивании абзаца.
POSTURE	Задает наклон шрифта.
RUN_DIRECTION	Задает направление просмотра текста
SIZE	Задает размер шрифта в пунктах.
STRIKETHROUGH	Задает перечеркивание шрифта.

Продолжение таблицы 5.4

Атрибут	Значение
SUPERSCRIPT	Задает подстрочные или надстрочные индексы.
SWAP_ COLORS	Заменяет местами цвет текста и цвет фона.
TRANSFORM	Задает преобразование шрифта.
UNDERLINE	Задает подчеркивание шрифта.
WEIGHT	Задает толщину шрифта.
WIDTH	Задает ширину шрифта.

Не все шрифты позволяют задавать все свои атрибуты. Посмотреть список допустимых атрибутов для данного шрифта можно методом `getAvailableAttributes()` класса `Font`.

В классе `Font` есть конструктор `Font(Map attributes)`, которым можно сразу задать нужные атрибуты создаваемому шрифту. Это требует предварительной записи атрибутов в специально созданный для этой цели объект класса, реализующего интерфейс `Map`: класса `HashMap`, `WeakHashMap` или `Hashtable`. Например:

```
HashMap hm = new HashMap();
hm.put(TextAttribute.SIZE, new Float(60.0f));
hm.put(TextAttribute.POSTURE,
TextAttribute.POSTUREJDBLIQUE);
Font f = new Font(hm);
```

Можно создать шрифт и другим конструктором: `Font(String name, int style, int size)`, а потом добавлять и изменять атрибуты методами `deriveFont()` класса `Font`.

Текст в `Java 2D` обладает собственным контекстом — объектом класса `FontRenderContext`, хранящим всю информацию, необходимую для вывода текста. Получить его можно методом `getFontRenderContext()` класса `Graphics2D`.

Вся информация о тексте, в том числе и об его контексте, аккумулируется в объекте класса `TextLayout`.

В конструкторе класса `TextLayout` задается текст, шрифт и контекст. Начало метода `paint()` со всеми этими определениями может выглядеть следующим образом:

```
public void paint(Graphics gr){
Graphics2D g = (Graphics2D)gr;
FontRenderContext frc = g.getFontRenderContext();
Font f = new Font("Serif", Font.BOLD, 15);
String s = "Какой-то текст";
TextLayout t1 = new TextLayout(s, f, frc);
// Продолжение метода }
```

В классе `TextLayout` есть метод, вычерчивающий содержимое объекта этого класса в графической области `g`, начиная с точки `(x, y)`:

`draw(Graphics2D g, float x, float y).`

Следующий метод возвращает контур шрифта в виде объекта `shape`: `getOutline(AffineTransform at)`. Этот контур можно затем заполнить по какому-нибудь образцу или вывести только контур

Пример, позволяющий вывести средствами Java 2.0 текст с тенью приведен ниже, а результат — на рисунке 5.5.

```
1  import java.awt.*;
2  import java.awt.font.*;
3  import java.awt.geom.*;
4  import java.awt.event.*;
5  class StillText extends Frame{ StillText(String
   s) {
6  super(s);
7  setSize(400, 200);
8  setVisible(true);
9  addWindowListener(new WindowAdapter(){
10 public void windowClosing(WindowEvent ev){
11 System.exit(0) ;
12 }
13 });
14 }
15 public void paint(Graphics gr){
16 Graphics2D g = (Graphics2D)gr;
17 int w = getSize().width, h = getSize().height;
18 FontRenderContext              frc              =
   g.getFontRenderContext();
```

```
19 String s = "Тень";
20 Font f = new Font(
21 "Serif", Font.BOLD, h/3);
22 TextLayout tl = new TextLayout(s, f, frc);
23 AffineTransform at = new AffineTransform();
24 at.setToTranslation(w/2-
   tl.getBounds().getWidth()/2, h/2);
25 Shape sh = tl.getOutline(at);
26 g.draw(sh);
27 AffineTransform atsh = new AffineTransform(1,
   0.0, 1.5, -1, 0.0, 0.0);
28 g.transform(at);
29 g.transform(atsh);
30 Font df = f.deriveFont(atsh);
31 TextLayout dtl = new TextLayout(s, df, frc);
32 Shape sh2 = dtl.getOutline(atsh);
33 g.fill(sh2);
34 }
35 public static void main(String[] args) {
36 new StillText("Эффект тени");
37 }
38 }
```

Контекст шрифта можно узнать после вызова метода `getFontRenderContext()` (строка 18). Далее создается строковая переменная `s`, где будут храниться значения выводимого текста (строка 19), шрифты `f` (строка 20) и `df` (строка 30) — копия шрифта `f` с аффинным преобразованием `atsh`. Затем создаются объекты класса `TextLayout` (строки 22, 31). Метод `setToTranslation(double tx, double ty)` задает преобразование для трансформации (строка 24). Происходит получение контура текста (строки 25, 32) — объектов класса `Shape sh` и `sh2`, а затем отображение этих фигур (строки 26, 33).

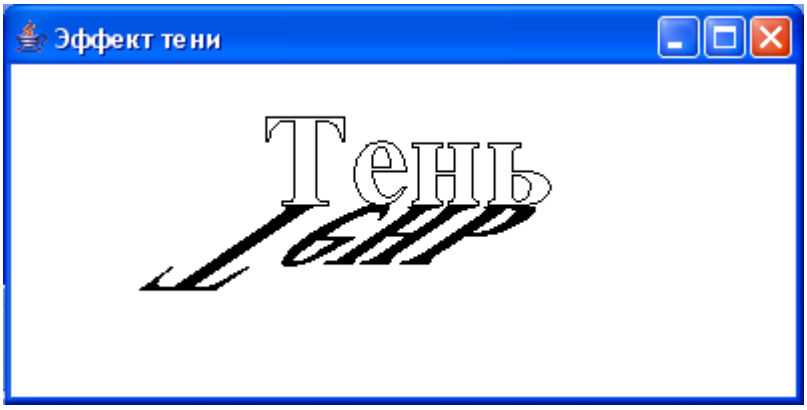


Рисунок 5.5 — Вывод текста средствами Java 2D

Еще одна возможность создать текст с атрибутами — определить объект класса `AttributedString` из пакета `java.text`. Конструктор этого класса `AttributedString(String text, Map attributes)` задает сразу и текст, и его атрибуты. Затем можно добавить или изменить характеристики текста одним их трех методов `addAttibute()`.

Если текст занимает несколько строк, то возникает проблема его форматирования по абзацам. Для этого вместо класса `TextLayout` используется класс `LineBreakMeasurer`, методы которого позволяют отформатировать абзац. Для каждого сегмента текста можно получить экземпляр класса `TextLayout` и вывести текст, используя его атрибуты.

При редактировании текста необходимо отслеживать текущую позицию курсора в тексте. Это осуществляется методами класса `TextHitInfo`, а методы класса `TextLayout` позволяют получить позицию курсора, выделить блок текста и подсветить его.

Можно задать отдельные правила для вывода каждого символа текста. Для этого надо получить экземпляр класса `GlyphVector` методом `createGlyphVector()` класса `Font`, изменить позицию символа методом `setGlyphPosition()`, задать преобразование символа, если это допустимо для данного шрифта, методом `setGlyphTransform()`, и вывести измененный текст методом `drawGlyphVector()` класса `Graphics2D`. Рассмотрим пример.

```
1   import java.awt.*;
```

```
2  import java.awt.font.*;
3  import java.awt.geom.*;
4  import java.awt.event.*;
5  class GlyphTest extends Frame{ GlyphTest(String
   s){
6  super(s) ;
7  setSize(400, 150);
8  setVisible(true);
9  addWindowListener(new WindowAdapter(){
10 public void windowClosing(WindowEvent ev){
11 System.exit(0);
12 }
13 });
14 }
15 public void paint(Graphics gr){
16 int h = 5;
17 Graphics2D g = (Graphics2D)gr;
18 FontRenderContext                          frc
   =g.getFontRenderContext();
19 Font f = new Font("Serif", Font.BOLD, 30);
20 GlyphVector   gv   =   f.createGlyphVector(frc,
   "Пляшущий текст");
21 int len = gv.getNumGlyphs();
22 for (int i = 0; i < len; i++){
23 Point2D.Double p=new Point2D.Double(25 * i, h=-h);
24 gv.setGlyphPosition(i, p) ;
25 }
26 g.drawGlyphVector(gv, 10, 100);
27 }
28 public static void main(String[] args){
29 new GlyphTest(" Вывод отдельных символов");
30 }
31 }
```

Для вывода отдельных символов в методе `paint()` (строка 15) необходимо получить настройки шрифта в `frc` (строка 18) и создать

шрифт `f` (строка 19). Затем получить экземпляр класса `GlyphVector` методом `createGlyphVector()` класса `Font`, передавая ему настройки шрифта `frc` и текст (строка 20). Длину надписи определить методом `getNumGlyphs()` класса `GlyphVector` (строка 21). В цикле (строка 22-25) происходит смещение позиции буквы методом `setGlyphPosition()` (строка 24). Затем осуществляется вывод текста методом `drawGlyphVector()` (строка 26).

Результат работы программы изображен на рисунке 5.6.

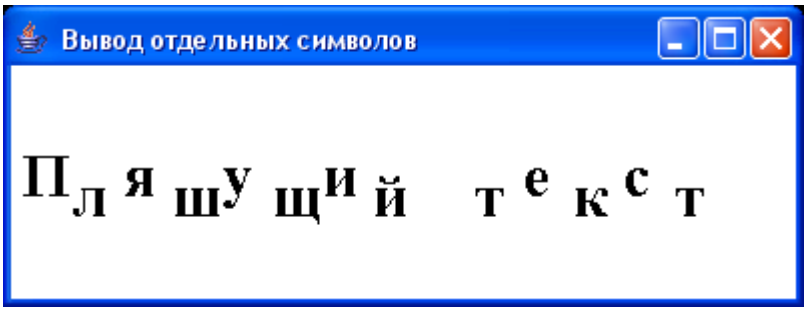


Рисунок 5.6 — Вывод отдельных символов

5.7 Пример построения графика

Одной из инженерных задач является графическое представление результатов экспериментов и испытаний. Одним из наиболее распространенных средств является график, построенный в декартовой системе координат.

Ниже приведен пример построения графика функции вида $f(x) = e^{\frac{-x}{2}} \sin(10x)$, а на рисунке 5.7 – результат работы программы.

```
1 import java.awt.*;
2 import java.awt.event.*;
3 import java.awt.geom.*;
4 import javax.swing.*;
5 import java.util.*;
6 class Graf2D
7 extends Frame {
8 Graf2D(String s) {
9 super(s);
10 setSize(new Dimension(600, 600));
```

```
11 setVisible(true);
12 addWindowListener(new WindowAdapter() {
13 public void windowClosing(WindowEvent ev) {
14 System.exit(0);
15 }
16 });
17 }
18 public void paint(Graphics g) {
19 Graphics2D g2 = (Graphics2D) g;
20 Dimension d = getSize();
21 float gridWidth = d.width * 0.7f, gridHeight =
d.height * 0.7f;//размеры области построения
//графика
22 float xn = 50, yn = 100;//начальные точки
области
//построения графика
23 double y;
24 //рисуем прямоугольник
25 g2.setPaint(Color.black);
26 GradientPaint gp = new GradientPaint(200, 200,
Color.white, 600, 600,
27 Color.green);
28 g2.draw(new Rectangle2D.Double(10, 40, d.width
- 20, d.height - 50));
29 g2.setPaint(gp);
30    g2.fill(new    Rectangle2D.Double(xn,    yn,
gridWidth, gridHeight));
31 g2.setPaint(Color.black);
32    g2.draw(new    Rectangle2D.Double(xn,    yn,
gridWidth, gridHeight));
33 //рисуем сетку
34 g2.setPaint(Color.lightGray);
35 float n = gridHeight / 10, m = gridWidth /
10;//размер сетки
36 for (int i = 1; i < 10; i++) {
```

```
37  g2.draw(new  Line2D.Double(xn,  n  *  i  +  yn,
gridWidth + xn, n * i + yn)); //горизонтальная
38 g2.draw(new Line2D.Double(m * i + xn, yn, m *
i + xn, gridHeight + yn)); //вертикальная
39 }
40 //оси
41 g2.setPaint(Color.black);
42 g2.setStroke(new BasicStroke(2));
43  g2.draw(new  Line2D.Double(xn,  n  *  5  +  yn,
gridWidth + xn, n * 5 + yn)); //горизонтальная
44   g2.draw(new   Line2D.Double(xn,   yn,   xn,
gridHeight + yn)); //вертикальная
45 //деления и подписи осей
46 float z;
47 Formatter fmt;
48 for (int i = 1; i < 10; i++) {
49 z = 1 - (float) i / 10 * 2;
50  g2.draw(new Line2D.Double(xn - 2.5, n * i +
yn, 2.5 + xn, n * i + yn)); //горизонтальная
51 fmt=new Formatter();
52 fmt.format("%4.1f",z);
53  g2.drawString(""+fmt, xn - 25, n * i + yn +
5);
54 g2.draw(new Line2D.Double(m * i + xn, yn - 2.5
+ 5 * n, m * i + xn,
55 2.5 + yn + 5 * n)); //вертикальная
56 g2.drawString("" + i, m * i + xn, 20 + yn + 5 *
n);
57 }
58 //подпись графика
59 g2.setPaint(new Color(50, 150, 240, 100));
60  g2.setFont(new  Font("Times  New  Roman",  3,
28));
61   g2.drawString("График   функции",   xn   +
gridWidth * 0.3f, yn - 30);
```

```
62 g2.setPaint(Color.black);
63 //подписи осей
64  g2.setFont(new  Font("Times  New  Roman",  3,
18));
65 g2.drawString("f(x)", xn - 30, yn + 5);
66 g2.drawString("x", gridWidth + xn + 10, yn +
gridHeight / 2 + 10);
67 //построение графика
68 float c = 2, k = 10;
69 g2.setStroke(new BasicStroke(1));
70 g2.setPaint(Color.red);
71 GeneralPath p = new GeneralPath();
72 p.moveTo(xn,(float) (Math.exp(0) * Math.sin(0)
* ( -gridHeight / 2) + n * 5 + yn));
73 for (float x = 0; x <= 10; x += 0.05)
74 {
75 y = Math.exp( -x / c) * Math.sin(k * x);
76   p.lineTo(x   *   gridWidth   /   10   +   xn,
(float) y * ( -gridHeight / 2) + n * 5 + yn);
77 g2.draw(p);
78 }
79 //легенда
80 g2.setStroke(new BasicStroke(1));
82 g2.setPaint(Color.gray);
82  g2.draw(new  Ellipse2D.Double(xn+gridWidth+10,
yn, d.width * 0.15f, 30));
83 g2.setStroke(new BasicStroke(3));
84 g2.setPaint(Color.red );
85   g2.draw(new   Line2D.Double(xn+gridWidth+20,
yn+15, xn+gridWidth+40, yn+15));
86 g2.setPaint(Color.black  );
87  g2.drawString("f(x)",  xn+gridWidth+50,  yn+20
);
88 }
89 public static void main(String[] args) {
```

```
90 new Graf2D("Пример графика");
91 }
92 }
```

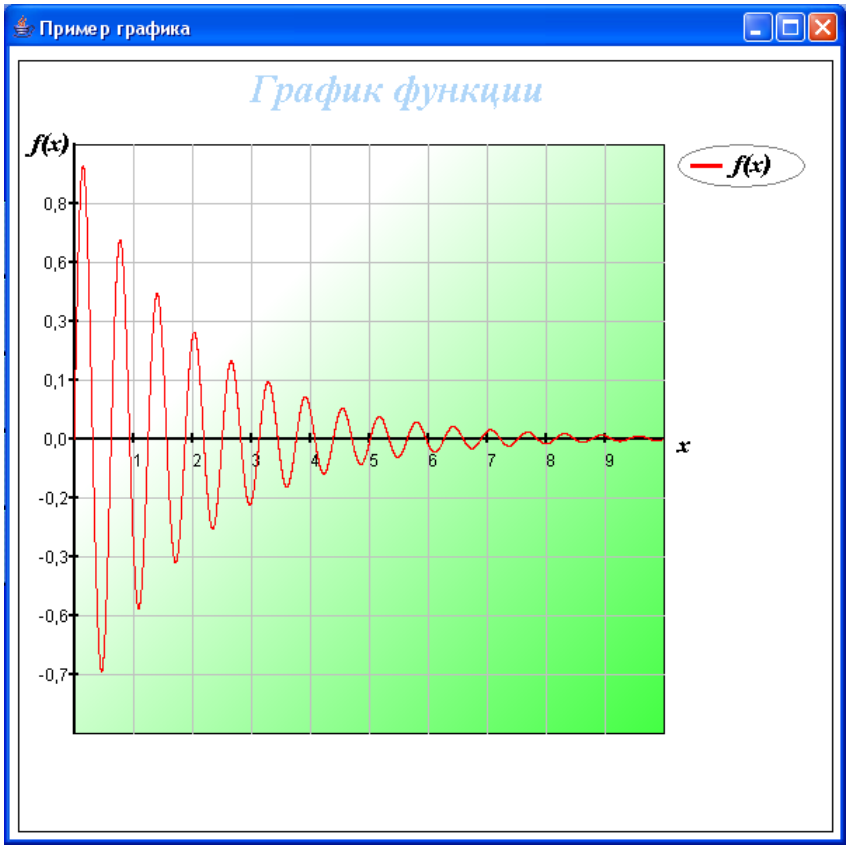


Рисунок 5.7 — График

В методе `Paint()` создается объект `Graphics2D g2` (строка 19). Далее определены переменные `gridWidth` и `gridHeight`, в которых будет храниться размер области построения (строка 21), и переменные `xn, yn` — начальные точки области построения (строка 22). Эти переменные будут использоваться для корректного построения изображения. Методом `setPaint()` задается текущий цвет графического контекста (строки 25, 29, 31, 34, 41, 59, 62, 70, 81, 84, 86). Методом `draw()` рисуются прямоугольники, ограничивающие область построения (строки 28, 32), линии сетки (строки 37,38), оси (строки 44,45), деления осей (строки 50,54), график (строка 77), легенда (строки 82,85). Для построения графика создан объект `p` класса `GeneralPath` (строка 71). Текущая точка методом `moveTo()`

перемещается в начало координат по оси абсцисс и в значение функции f(0) (строка 72). В цикле методом `lineTo()` добавляются к объекту `p` отрезки линий с координатами от текущей точки до точки `(x, f(x))` (строка 76), затем происходит отображение этого объекта методом `draw()` (строка 77). Значение `x` изменяется в диапазоне от 0 до 10 с шагом 0,05. Значение координат для выводимых отрезков корректируются с учетом масштаба и начальных точек области построения графика.

Шрифты устанавливаются в строках 60 и 64; вывод текста осуществляется методом `drawString()`: подписи делений (строки 53, 56), графика (строка 61), осей (строки 65, 66), легенды (строка 87).

Для форматированного вывода подписей по осям объявлена переменная `fmt` типа `Formatter` (строка 47). В цикле, в котором выводятся оси, переопределяется объект `Formatter` (строка 51) и задается его формат методом `format()`, в качестве параметров передается формат `(%КолВыводимыхСимволов.КолЗнаковПослеЗапятой)` и форматируется переменная (строка 52).

5.8 Задание к лабораторной работе

1. Напишите программу, которая отображает график заданной функции (табл. 5.5). На графике отобразите название и подписи осей, оформленные различными шрифтами, а также линии сетки, подписи значений по шкале X и Y. При оформлении используйте градиентную заливку (рис. 5.4).

2. Создайте столбцовую диаграмму выпуска продукции цехами (таблица 5.6), используя метод `draw3Drect()`.

3. Распечатайте результаты и тексты программ.

Таблица 5.5 — Функции для задания № 1

№ вар.	Функция	Диапазон и шаг изменения переменной x
1	$f(x) = 20 \cdot e^{-\frac{x}{10}} \cdot \sin(x)$	$x \in [0; 6\pi] \quad \Delta x = \pi / 36$
2	$f(x) = \frac{\cos^2(x) \cdot \ln(x)}{5}$	$x \in \left[\frac{\pi}{2}; 6\pi\right] \quad \Delta x = \pi / 36$

Продолжение таблицы 5.5

№ вар.	Функция	Диапазон и шаг изменения переменной x
3	$f(x)=\dfrac{e^{-\frac{x}{5}}\cdot\cos\left(\frac{x}{2}\right)}{2}$	$x\in[0;20]\quad \Delta x=0.1$
4	$f(x)=e^{\sin 2x}-e^{\cos 1.2x}$	$x\in[0;6\pi]\quad \Delta x=0.25$
5	$f(x)=\dfrac{\lg^2(x)+\ln^3(x)}{e^{0.25x}}$	$x\in\left[\dfrac{\pi}{2};20\pi\right]\quad \Delta x=\pi/10$
6	$f(x)=\cos^2(x)+\sin^3(x)$	$x\in[0;6\pi]\quad \Delta x=\pi/36$
7	$f(x)=\cos^3(x)+\sin^2(x)$	$x\in[0;6\pi]\quad \Delta x=\pi/36$
8	$f(x)=2\cos^3(x)+3\sin^2(x)$	$x\in[0;6\pi]\quad \Delta x=\pi/36$
9	$f(x)=\dfrac{\sin(x)}{2}-\dfrac{\cos(x)}{3}e^x$	$x\in\left[\dfrac{\pi}{2};10\pi\right]\quad \Delta x=\pi/10$
10	$f(x)=\dfrac{\cos^{2.3}\left(\frac{x}{2}\right)\cdot\ln\left(x^{0.67}\right)}{5}$	$x\in\left[\dfrac{\pi}{2};10\pi\right]\quad \Delta x=\pi/12$
11	$f(x)=\dfrac{e^{-\frac{8x}{7}}}{x^2}\cdot\cos\dfrac{\sqrt{x^5}}{3}$	$x\in[2;6]\quad \Delta x=0.1$
12	$f(x)=\dfrac{e^{-2x}}{x}\cdot\sin x^2$	$x\in[2;5]\quad \Delta x=0.1$
13	$f(x)=\dfrac{e^{-\frac{2x}{5}}}{x^{1.1}}\cdot\sin\dfrac{x^2}{3}$	$x\in[2;10]\quad \Delta x=0.1$
14	$f(x)=e^{\frac{x}{2}}+10\sin^2 x$	$x\in[0;5]\quad \Delta x=0.1$
15	$f(x)=e^{\frac{3x}{2}}+15\sin^2 x^3$	$x\in[0;2]\quad \Delta x=0.2$
16	$f(x)=2.5\cdot x\cdot e^{-0.2x}\cdot\sin\dfrac{x}{\pi}$	$x\in[0;20\pi]\quad \Delta x=\pi/10$
17	$f(x)=2.2\cos^{\frac{2}{3}}x\cdot e^{-0.25x}$	$x\in[0;50]\quad \Delta x=1$
18	$f(x)=\cos^2 x\cdot e^{-\frac{x}{2}}$	$x\in[0;5]\quad \Delta x=0.1$

Продолжение таблицы 5.5

№ вар.	Функция	Диапазон и шаг изменения переменной x
19	$f(x) = \sin x^2 \cdot e^{-\frac{2x}{3}}$	$x \in [0;2]$ $\Delta x = 0.1$
20	$f(x) = \cos x \cdot e^{\frac{x}{2}} \cdot \sin x^2$	$x \in [3;5]$ $\Delta x = 0.05$
21	$f(x) = x \cdot e^{-0.8x} \cdot \cos\sqrt{x^3}$	$x \in [1;5]$ $\Delta x = 0.05$
22	$f(x) = \sin^2 x^3 + e^{-x} \cdot \cos\sqrt{x}$	$x \in [0;2]$ $\Delta x = 0.05$
23	$f(x) = e^{-\frac{x}{3}} \cdot \cos\frac{x^2}{3} \cdot tg\sqrt[5]{x}$	$x \in [0;5]$ $\Delta x = 0.05$
24	$f(x) = e^{-\frac{2x}{3}} \cdot \sin\frac{x^2}{3} \cdot \sqrt[3]{x}$	$x \in [0;5]$ $\Delta x = 0.05$
25	$f(x) = \cos^2(x^3) - \sin^3(x^2)$	$x \in [0;6\pi]$ $\Delta x = \pi / 36$

Таблица 5.6 — Исходные данные для задания № 2

Месяц	Цех1	Цех2	Цех3
1	120·N	$201 \cdot \frac{N}{N+1}$	$56 \cdot \frac{N}{N+1}$
2	150·N	$196 \cdot \frac{N}{N+1}$	$50 \cdot \frac{N}{N+1}$
3	111·N	$220 \cdot \frac{N}{N+1}$	$48 \cdot \frac{N}{N+1}$
4	125·N	$231 \cdot \frac{N}{N+1}$	$52 \cdot \frac{N}{N+1}$
5	180·N	$190 \cdot \frac{N}{N+1}$	$51 \cdot \frac{N}{N+1}$
6	154·N	$190 \cdot \frac{N}{N+1}$	$53 \cdot \frac{N}{N+1}$
7	130·N	$200 \cdot \frac{N}{N+1}$	$53 \cdot \frac{N+2}{N+1}$
8	150·N	$212 \cdot \frac{N}{N+1}$	$57 \cdot \frac{N+2}{N+1}$

Продолжение таблицы 5.6

Месяц	Цех1	Цех2	Цех3
9	165·N	$211\cdot\frac{N}{N+1}$	$60\cdot\frac{N+2}{N+1}$
10	143·N	$215\cdot\frac{N}{N+1}$	$49\cdot\frac{N+2}{N+1}$
11	127·N	$200\cdot\frac{N}{N+1}$	$50\cdot\frac{N+2}{N+1}$
12	142·N	$198\cdot\frac{N}{N+1}$	$50\cdot\frac{N+2}{N+1}$

N — номер варианта (номер по списку).

5.9 Контрольные вопросы

1. Контекст отображения Java 2D.

2. Методы для установки атрибутов в контексте отображения Graphics2D.

3. Приведите примеры, как установить текущий цвет, начертания линий, шрифт.

4. Общие методы отображение в Graphics2D.

5. Как отобразить контур геометрической фигуры. Приведите примеры.

6. Как отобразить закрашенную геометрическую фигуру. Приведите примеры.

7. Какие Вы знаете классы для построения графических примитивов.

8. Конструкторы для построения линий. Пример.

9. Конструкторы для построения прямоугольников. Пример.

10. Конструкторы для построения дуг. Пример.

11. Конструкторы для построения эллипсов и окружностей. Пример.

12. Класс GeneralPath. Назначение, конструкторы.

13. Как изменить текущую точку объекта GeneralPath.

14. Как провести отрезок прямой от текущей точки к точке (x, y).

15. Класс BasicStroke. Назначение, конструкторы, пример.

16. Как начертить штрихпунктирую линию.

17. Как изменить толщину и оформление конца линии.

18. Как нарисовать закрашенный сплошным цветом прямоугольник, как изменить его прозрачность.

19. Класс GradientPaint. Назначение, конструкторы, пример.

20. Как нарисовать окружность, закрашенную красно-желтой градиентной заливкой.

21. Как сделать, чтобы градиент несколько раз повторялся в рамках одной фигуры.

22. Класс TexturePaint. Назначение, конструкторы.

23. Как нарисовать эллипс, закрашенный текстурной заливкой.

24. Класс Font. Назначение, конструкторы, пример.

25. Как создать наклонный шрифт размером 16 пт.

26. Как создать тень у шрифта.

27. Как создать закрашенный градиентной заливкой шрифт.

28. Класс TextLayout. Назначение, конструкторы, пример.

ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет оформляется на стандартных листах белой бумаги формата А4 (210х297 мм). Текст пишется с одной стороны листа чернилами синего или черного цвета или печатается на принтере.

Бланк отчета готовится во время домашней подготовки к работе и должен содержать:

– данные о студенте: фамилия и инициалы, шифр группы;

– тему и цель работы;

– дату выполнения работы;

– набор данных, необходимых для выполнения работы: задание к работе, код программы, краткие теоретические сведения о типах данных и операциях, используемых классах.

После выполнения работы к отчету подшиваются: распечатка программного кода, распечатка результатов.

Отчет должен быть выполнен аккуратно. Неаккуратно выполненные отчеты, а также ксерокопии чужих отчетов или их фрагментов к защите не допускаются.

СПИСОК РЕКОМЕНДОВАННОЙ ЛИТЕРАТУРЫ

1. Савитч Уолтер. Язык Java. Курс программирования : пер. с англ. — 2-е изд. – М. : Издательский дом «Вильямс», 2002. — 928 с.; ил.

2. Калверт Ч. JBuilder: Разработка профессиональных приложений : пер. с англ. / Ч. Калверт, М. Калверт. — К. : ООО «ТИД «ДС»», 2004. — 1008 с.

3. Портянкин И. Swing. Эффектные пользовательские интерфейсы / И. Портянкин. — М. : Питер, 2005. — 528 с.

4. Хорстманн К. С. Библиотека профессионала. Java 2. Том 1. Основы : пер. с англ. / К.С. Хорстманн, Г. Корнелл. — М. : Издательский дом "Вильямс", 2014. — 864 с.; ил.

5. Gosling J. The Java Language Specification / James Gosling, Bill Jo, Guy Steele. — Boston : Addison-Wesley Longman Publishing Co., 1996. — p. 852.

6. Гэри Д. Java Server Pages. Библиотека профессионала : пер. с англ. / Дэвид М. Гери. — М. : Издательский дом "Вильямс", 2002. — 432 с.

7. Хокинс С. Администрирование Web-сервера Apache и руководство по электронной коммерции : пер. с англ. / Скотт Хокинс. — М. : Издательский дом «Вильямс», 2001. — 336 с.; ил.

ПРИЛОЖЕНИЕ А

```
import javax.swing.*;
import java.awt.*;
public class AllLayoutsDemo extends JFrame{
    public static final int WIDTH = 300;
    public static final int HEIGHT = 200;
  public static void main(String[] args){
   AllLayoutsDemo ald = new AllLayoutsDemo();
   ald.setVisible(true); }
public AllLayoutsDemo()  { // Объявление класса
  setSize(WIDTH, HEIGHT);
  addWindowListener(new WindowDestroyer());
  setTitle("Демонстрация использования всех видов
компоновки");
  JPanel contentPane;
BorderLayout borderLayout1 = new BorderLayout();
JPanel jPanel1 = new JPanel();
JPanel jPanel2 = new JPanel();
JPanel jPanel3 = new JPanel();
JPanel jPanel4 = new JPanel();
JPanel jPanel5 = new JPanel();
JPanel jPanel6 = new JPanel();
GridLayout gridLayout1 = new GridLayout();
JButton jButton1 = new JButton();
JButton jButton2 = new JButton();
JButton jButton3 = new JButton();
JButton jButton4 = new JButton();
JButton jButton5 = new JButton();
JButton jButton6 = new JButton();
JButton jButton7 = new JButton();
JLabel jLabel1 = new JLabel();
JTextField jTextField1 = new JTextField();
GridLayout gridLayout2 = new GridLayout();
JRadioButton jRadioButton1 = new JRadioButton();
JRadioButton jRadioButton2 = new JRadioButton();
```

```
JRadioButton jRadioButton3 = new JRadioButton();
JRadioButton jRadioButton4 = new JRadioButton();
JRadioButton jRadioButton5 = new JRadioButton();
JCheckBox jCheckBox1 = new JCheckBox();
JCheckBox jCheckBox2 = new JCheckBox();
JCheckBox jCheckBox3 = new JCheckBox();
GridBagLayout gridBagLayout1 =
              new GridBagLayout();
JLabel jLabel2 = new JLabel();
JLabel jLabel3 = new JLabel();
contentPane = (JPanel) this.getContentPane();
contentPane.setLayout(borderLayout1);
this.setSize(new Dimension(600, 300));
this.setTitle("Использование разных компоновок");
jPanel1.setBorder(BorderFactory.createLoweredBevelB
order());
jPanel2.setBorder(BorderFactory.createLoweredBevelB
order());
jPanel2.setMaximumSize(new Dimension(180, 37));
jPanel2.setMinimumSize(new Dimension(180, 37));
jPanel2.setLayout(gridBagLayout1);
jPanel3.setBorder(BorderFactory.createLoweredBevelB
order());
jPanel3.setDebugGraphicsOptions(0);
jPanel3.setLayout(gridLayout2);
jPanel5.setBorder(BorderFactory.createLoweredBevelB
order());
jPanel5.setLayout(gridLayout1);
jPanel6.setBorder(BorderFactory.createEtchedBorder(
));
jPanel6.setSize(100,100);
jButton1.setText("jButton1");
gridLayout1.setColumns(2);
gridLayout1.setRows(2);
```

```
jButton2.setText("jButton2");
jButton3.setText("jButton3");
jButton4.setText("jButton4");
jButton5.setBackground(Color.magenta);
jButton5.setText("jButton5");
jButton6.setText("jButton6");
jButton7.setText("jButton7");
jLabel1.setText("Панель 1");
jTextField1.setText("Панель 4");
jPanel4.setBackground(Color.yellow);
jRadioButton1.setText("jRadioButton1");
jRadioButton2.setSelected(true);
jRadioButton2.setText("jRadioButton2");
gridLayout2.setColumns(2);
gridLayout2.setRows(3);
jRadioButton3.setText("jRadioButton3");
jRadioButton4.setText("jRadioButton4");
jRadioButton5.setText("jRadioButton5");
jCheckBox1.setText("jCheckBox1");
jCheckBox2.setSelected(true);
jCheckBox2.setText("jCheckBox2");
jCheckBox3.setText("jCheckBox3");
jLabel2.setText("Панель 2");
jLabel3.setText("Панель 3");
contentPane.add(jPanel1, BorderLayout.NORTH);
jPanel1.add(jLabel1, null);
contentPane.add(jPanel2, BorderLayout.WEST);
contentPane.add(jPanel3, BorderLayout.SOUTH);
jPanel3.add(jLabel3, null);
jPanel3.add(jRadioButton1, null);
jPanel3.add(jRadioButton2, null);
jPanel3.add(jRadioButton3, null);
jPanel3.add(jRadioButton4, null);
jPanel3.add(jRadioButton5, null);
contentPane.add(jPanel4, BorderLayout.EAST);
```

```
jPanel4.add(jTextField1, null);
contentPane.add(jPanel5, BorderLayout.CENTER);
jPanel5.add(jPanel6, null);
jPanel6.add(jButton1, null);
jPanel6.add(jButton2, null);
jPanel6.add(jButton3, null);
jPanel6.add(jButton4, null);
jPanel5.add(jButton5, null);
jPanel5.add(jButton6, null);
jPanel5.add(jButton7, null);
jPanel2.add(jCheckBox1, new
GridBagConstraints(0, 1, 1, 1, 0.0, 0.0,
GridBagConstraints.SOUTHWEST,
GridBagConstraints.NONE, new Insets(0, 0, 0, 12),
16, -1));
jPanel2.add(jCheckBox2, new GridBagConstraints (0,
2,   1,   1,   0.0,   0.0,GridBagConstraints.WEST,
GridBagConstraints.NONE, new Insets(0, 2, 0, 20),
26, 2));
jPanel2.add(jCheckBox3, new GridBagConstraints (0,
3, 1, 1, 0.0, 0.0,
GridBagConstraints.NORTHWEST,GridBagConstraints.NON
E,
new Insets(-1, 3, 6, 14), 13, 2));
jPanel2.add(jLabel2,  new GridBagConstraints(0, 0,
1,   1,   0.0,   0.0,   GridBagConstraints.CENTER,
GridBagConstraints.NONE, new Insets(0, 0, 0, 0), 0,
0)); }
}
```

ПРИЛОЖЕНИЕ Б

Компонент JTextPane

Компонент `JEditorPane` используется в основном для статического отображения уже созданных текстовых документов некоторого формата (заданного классом `EditorKit`), а унаследованный от него текстовый компонент `JTextPane` незаменим при создании в приложении многофункционального текстового редактора. Обладая всеми возможностями своего предка `JEditorPane`, класс `JTextPane` добавляет к нему разметку текста стилями. Для этого в нем используется специальная модель документа `StyledDocument` и настроенная на поддержку такой модели фабрика классов `StyledEditorKit`.

Концепция стиля в текстовом редакторе многократно повышает эффективность работы пользователя с текстом. Стиль (style) – это некоторый набор атрибутов редактируемого текста (такими атрибутами могут быть шрифт текста, его размер и цвет, выравнивание и т. п.), который можно применить к любому фрагменту текста. Стиль позволяет четко разделить внешний вид документа и собственно сам текст. Пользователь может сосредоточиться на наборе текста, используя при этом несколько стилей (пара заголовков, основной текст, текст сносок), а если у него возникнет необходимость оформить текст по-другому, понадобится лишь поменять атрибуты стилей. Текст, набранный с использованием стилей, изменится автоматически. Стили можно найти в любом современном редакторе, в том числе они встроены в язык разметки HTML посредством расширений CSS (Cascaded Style Sheets — каскадные таблицы стилей).

Помимо стилей (в компоненте `JTextPane` стили идентифицируются строковыми именами) поддерживаются и просто неупорядоченные наборы атрибутов текста, заданные объектами `AttributeSet`. Наборы атрибутов позволяют изменять произвольные фрагменты текста или даже целые абзацы, слегка отступая от комплекта заранее заданных стилей. Это также очень полезная возможность, хотя всегда лучше создавать на каждый отдельный формат текста свой уникальный стиль.

Одной из самых полезных является способность стилей образовывать иерархии. Одни стили могут наследовать атрибуты других стилей, прибавляя при этом что-то свое. Например, стиль для основного текста может определять шрифт, его размер и выравнивание, а стиль для заголовка, унаследованный от этого базового стиля, может поменять только размер шрифта, не меняя остальное. Иерархия стилей дает пользователю возможность еще быстрее настраивать внешний вид и формат документа, не затрагивая текст: сменив шрифт в основном стиле, пользователь сменит его и для всех унаследованных от него стилей.

В коде ниже представлен пример работы с компонентом JTextPan.

```
1   import javax.swing.*;
2   import javax.swing.text.*;
3   import java.awt.Color;
4   import java.awt.event.*;
5   public class StyledText extends JFrame {
6   private JTextPane textPane;
7     public StyledText() {
8       super("Текстовый редактор");
9   setDefaultCloseOperation(EXIT_ON_CLOSE);
10  // создадим объект-редактор
11  textPane = new JTextPane();
12  // создание документа и стилей
13    createDocument(textPane);
14  // добавим редактор в панель содержимого
15  getContentPane().add(
                  new JScrollPane(textPane));
16  // выводим окно на экран
17  setSize(400, 300); setVisible(true);
18                        }
19  private void createDocument(JTextPane tp) {
20  // настроим стили
21  // стиль основного текста
22  Style normal = tp.addStyle("Normal", null);
```

```
23  StyleConstants.setFontFamily(normal,
    "Verdana");
24  StyleConstants.setFontSize(normal, 13);
25  // заголовок
26  Style   heading   =   tp.addStyle("Heading",
    normal);
27  StyleConstants.setFontSize(heading, 20);
28  StyleConstants.setBold(heading, true);
29  // пишем в документ текст, используя стили
30  insertString("Простой     Заголовок",     tp,
    heading);
31  insertString("Обычное     содержимое,",     tp,
    normal);
32  insertString("использующее стиль Normal.", tp,
    normal);
33  insertString("Еще   Один   Заголовок",   tp,
    heading);
34  // меняем произвольную часть текста
35  SimpleAttributeSet      red      =      new
    SimpleAttributeSet();
36  StyleConstants.setForeground(red, Color.red);
37  StyledDocument doc = tp.getStyledDocument();
38  doc.setCharacterAttributes(5, 5, red, false);
39  // добавим компонент в конец текста
40  tp.setCaretPosition(doc.getLength());
41  JCheckBox  check  =  new  JCheckBox("Флажок  в
    тексте!");
42  check.setOpaque(false);
43  tp.insertComponent(check);
44  }
45  // вставим строку в конец документа с
46  // переносом, используя стиль оформления
47  private void insertString(String s, JTextPane
    tp, Style style) {
48  try { // попробуем получить данные
```

```
49  Document doc = tp.getDocument();
50  doc.insertString(doc.getLength(),s  +  "\r\n",
    style);
51  } catch (Exception ex) {
52      ex.printStackTrace();
53  }
54  }
55  public static void main(String[] args) {
56  new StyledText();           }
57  }
```

В примере создается текстовый редактор `JTextPane`, который размещается в центре панели содержимого окна с рамкой (редактор предварительно размещен в панели прокрутки `JScrollPane`). Основные действия происходят в методе `createDocument()`, в котором создаются стили и документ наполняется содержимым.

Сначала создаются два именованных стиля, которые затем будут использованы при разметке текста. Создается базовый для всего документа стиль (с названием Normal). Метод `addStyle()` применяется для добавления стиля в редактор. Ему нужно передать два параметра: название нового стиля и стиль, который будет являться родительским для нового стиля, а в ответ он вернет новый стиль `Style`. Если в качестве второго параметра передать пустую ссылку `null`, стиль окажется без родителя и будет создаваться «с нуля». Базовый стиль своего документа создается именно так. Второй стиль в документе из примера (с названием Heading) служит для заголовков. Он унаследован от стиля Normal, а значит, перенимает от своего родителя все установленные для того атрибуты, такие как размер и цвет шрифта. Стиль заголовка отличается увеличенным размером шрифта и полужирным начертанием. Обратите внимание, что для установки атрибутов стилей используются удобные статические методы класса `StyleConstants`.

Такой же метод применяется и в модели `StyledDocument`. На самом деле практически все методы, определенные в классе `JTextPane` и так или иначе манипулирующие текстом, его атрибутами

и стилями, определены в модели документа StyledDocument. В больших приложениях лучше работать с моделью напрямую.

После создания стилей можно приступать к вставке в документ текста, размеченного только что настроенными стилями. Вставить текст в компонент JTextPane можно только посредством модели документа Document, в которой имеется метод insertString(). Этот метод требует три параметра: позицию для вставки, строку, которую необходимо вставить, и стиль, который будет иметь вставляемая строка. В программе вставкой текста в документ занимается метод insertString(). В нем строка добавляется к концу документа и при этом дополняется символами переноса строки (их приходится указывать вручную, автоматически они не добавляются). Для вставки текста в конец документа в качестве первого параметра метода insertString() необходимо указать размер документа, узнать его можно с помощью метода getLength(). В программе приходится следить за исключением, которое возникнет в том случае, если позиция в документе будет указана неверно. С помощью метода insertString()в документ добавляется текст: сначала заголовок, потом — несколько строк с обычным стилем, и снова заголовок (используя созданные стили).

Далее демонстрируется, как можно изменять оформление произвольного фрагмента текста, каким бы стилем он ни был размечен. Для этого предназначен метод setCharacterAttributes(), которому необходимо указать диапазон в тексте, набор атрибутов, а также булево значение. Последнее указывает, нужно ли полностью заменить имеющийся стиль новым набором атрибутов или надо совместить имеющийся стиль с новым набором. Для хранения набора атрибутов применяется класс SimpleAttributeSet1, в котором указывается, что текст должен иметь красный цвет. Передав этот набор в метод и, отказавшись от полной замены имеющегося стиля (третий параметр равен false), часть заголовка закрашивается в красный цвет, не меняя остальных его атрибутов, подобных размеру и шрифту.

И наконец, используются некоторые специфические возможности JTextPane: добавляется в конец документа компонент — флажок

JCheckBox. Предварительно флажок делается прозрачным (свойство opaque устанавливается в false), в противном случае он будет закрашивать свою область цветом фона и выпадать из общей картины документа. Также необходимо переместить курсор в конец документа методом setCaretPosition(), иначе флажок появится вместо конца документа в его начале. Ну а сама вставка компонента на текущую позицию (указываемую курсором) выполняется методом insertComponent().

Текстовый редактор JTextPane незаменим там, где нужны многофункциональные возможности обработки текста. С его помощью можно управлять стилями, обновлять произвольные фрагменты текста, вставлять в текст компоненты и значки, и т.д.

Набор атрибутов описывается интерфейсом AttributeSet, класс SimpleAttributeSet — простейшая реализация этого интерфейса, позволяющая быстро настроить набор. Стили также являются наборами атрибутов: класс стилей Style реализует интерфейс AttributeSet, отличие лишь в том, что стиль обладает собственным уникальным именем.

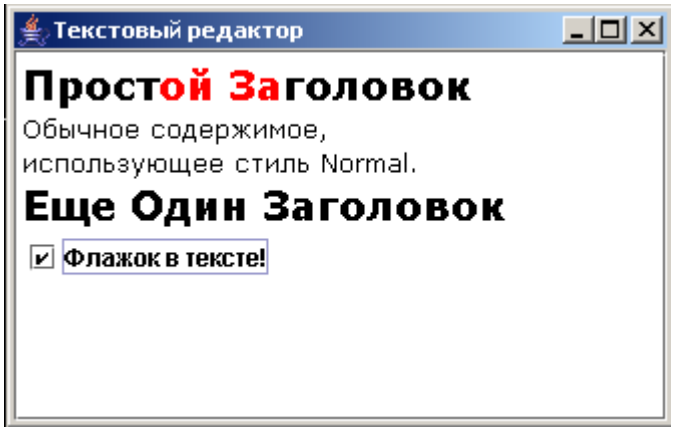


Рисунок Б.1 — Пример использования компонента JTextPane

ПРИЛОЖЕНИЕ В

Основы работы с файлами в Java

Файл — это именованная последовательность байт, хранящихся на диске. В Java для файлового ввода/вывода создаются байтовые потоки с помощью классов `FileInputStream` и `FileOutputStream`.

При чтении или записи текстовых файлов нужно организовать преобразование байтов в символы и обратно. Чтобы облегчить это преобразование, в пакет `java.io` введены классы `FileReader` и `FileWriter`. Они организуют преобразование потока: со стороны программы потоки символьные, со стороны файла — байтовые. Это происходит потому, что данные классы расширяют классы `InputStreamReader` и `OutputStreamWriter`, и соответственно, содержат методы преобразования внутри себя.

В конструкторах всех четырех указанных выше файловых потоков задается имя файла в виде строки типа `String` или же ссылка на объект класса `File`. Конструкторы не только создают объект, но и отыскивают файл и открывают его, например:

```
FileInputStream fs = new
FileInputStream("Example.Java");
FileReader fr = new
FileReader("C:\\Temp\\Example.Java ");
```

При неудаче создается исключение класса `FileNotFoundException`, но конструктор класса `FileWriter` создает также более общее исключение `IOException`.

После открытия выходного потока типа `FileWriter` или `FileOutputStream` содержимое файла стирается. Для того чтобы можно было делать запись в конец файла, и в том и в другом классе предусмотрен конструктор с двумя аргументами. Если второй аргумент равен `true`, то происходит запись в конец файла, если `false`, то файл заполняется новой информацией, Например:

```
    FileWriter fw = new FileWriter("ExAdd.txt",
true);
```

```
    FileOutputstream fos = new
FileOutputstream("C:\\Temp\\newfile.txt");
```

Сразу после выполнения конструктора можно читать файл:

```
fs.read(); fr.read();
```

или записывать в него:

```
fs.write((char)c); fw.write((char)c);
```

По окончании работы с файлом поток следует закрыть методом `close()`.

Преобразование потоков в классах `FileReader` и `FileWriter` выполняется по кодовым таблицам. Для правильного ввода кириллицы следует применять `FileReader`, а не `FileInputStream`. Если файл содержит текст в кодировке, отличной от локальной кодировки, то необходимо добавлять преобразование «вручную», например так:

```
InputStreamReader isr =
new InputStreamReader(fs, "KOI8_R"));
```

При работе с `RTFEditorKit` работу с файлом можно построить таким образом:

```
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import javax.swing.JEditorPane;
import javax.swing.JFrame;
public class RTF_EditorPane  {
public RTF_EditorPane(){
    JEditorPane editorPane = new JEditorPane();
    editorPane.setContentType("text/rtf");
    // Чтение из файла
    try   {
InputStreamReader isr = new InputStreamReader(
new FileInputStream("C:\\Temp\\testInp.rtf"),"windo
ws-1251");
BufferedReader in = new BufferedReader(isr);
editorPane.read(in, null);
```

```
    }
    catch (FileNotFoundException e) {}
    catch (IOException e)           {}
. . . . . .
. . . . . .
// Запись в файл
    try   {
OutputStreamWriter osr = new OutputStreamWriter(
      new    FileOutputStream("C:\\Temp\\testOut.rtf
"),"windows-1251");
    BufferedWriter out = new BufferedWriter(osr);
    editorPane.write(out);
}
catch (FileNotFoundException e1) {}
catch (IOException e2)           {}
. . . . . .
}
public static void main(String[] args)
{
    RTF_EditorPane rtf_EditorPane =
new RTF_EditorPane();
}
}
```

УЧЕБНОЕ ИЗДАНИЕ

Евгений Евгеньевич Бизянов
Артур Альбертович Гутник

ПРОГРАММИРОВАНИЕ

Лабораторный практикум

В авторской редакции

Художественное оформление обложки Н. В. Чернышова

Заказ № 295. Формат 60x84 $^{1}/_{16}$.
Бумага офс. Печать RISO.
Усл. печат. л. 11,1 Уч.-изд. л. 9,6
Издательство не несет ответственность за содержание
материала, предоставленного автором к печати.
Издатель и изготовитель:
ГОУ ВПО ЛНР «Донбасский государственный технический университет»
пр. Ленина, 16, г. Алчевск, ЛНР, 94204
(ИЗДАТЕЛЬСКО-ПОЛИГРАФИЧЕСКИЙ ЦЕНТР, ауд. 2113, т/факс 2-58-59)
Свидетельство о государственной регистрации издателя, изготовителя
и распространителя средства массовой информации
МИ-СГР ИД 000055 от 05.02.2016