Министерство образования Республики Беларусь

Учреждение образования  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Кафедра информационных систем управления

ОТЧЕТ

По дисциплине:

«Учебная практика»

Выполнил

студент гр. 9

Кравцова В.А.

Проверил

Высоких Л.К.

Минск 2016

Оглавление

[**ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №1** 3](#_Toc452459446)

[**ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №2** 5](#_Toc452459455)

[**ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №3** 7](#_Toc452459465)

[**ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №4** 9](#_Toc452459474)

[**ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №5** 11](#_Toc452459483)

[**ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №6** 13](#_Toc452459492)

[**ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №7** 15](#_Toc452459500)

[**ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №8** 17](#_Toc452459510)

[**ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №9** 20](#_Toc452459519)

[**ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №10** 23](#_Toc452459527)

[**ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №11** 25](#_Toc452459537)

[**ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №12** 28](#_Toc452459546)

[**ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №13** 30](#_Toc452459554)

**ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №1**

***Цель:***

Научиться использовать средства двумерной графики в Java из пакета java.awt, в том числе научится создавать собственные классы фигур, реализующих интерфейс Shape, и использовать класс AffineTransform для их трансформирования.

***Предметная область:***

Двумерная графика.

1. **Краткие теоретические сведения**

Средства двумерной графики в Java включают в себя следующее:

* Класс Graphics2D, который отвечает за отрисовку.
* Интерфейс Shape для фигур.
* Интерфейс Stroke, описывающий отрисовку линий.
* Интерфейс Paint, описывающий заливку фигур.
* Интерфейс Composite, описывающий взаимодействие цветов с нижележащими слоями.
* Класс RenderingHints для настройки параметров отрисовки как, например, сглаживание.
* Класс AffineTransform, определяющий аффинные преобразования.

1. **Задание**
   1. ***Постановка задачи:***

Изобразить эллипс, вращающийся в плоскости экрана вокруг своего центра против часовой стрелки.

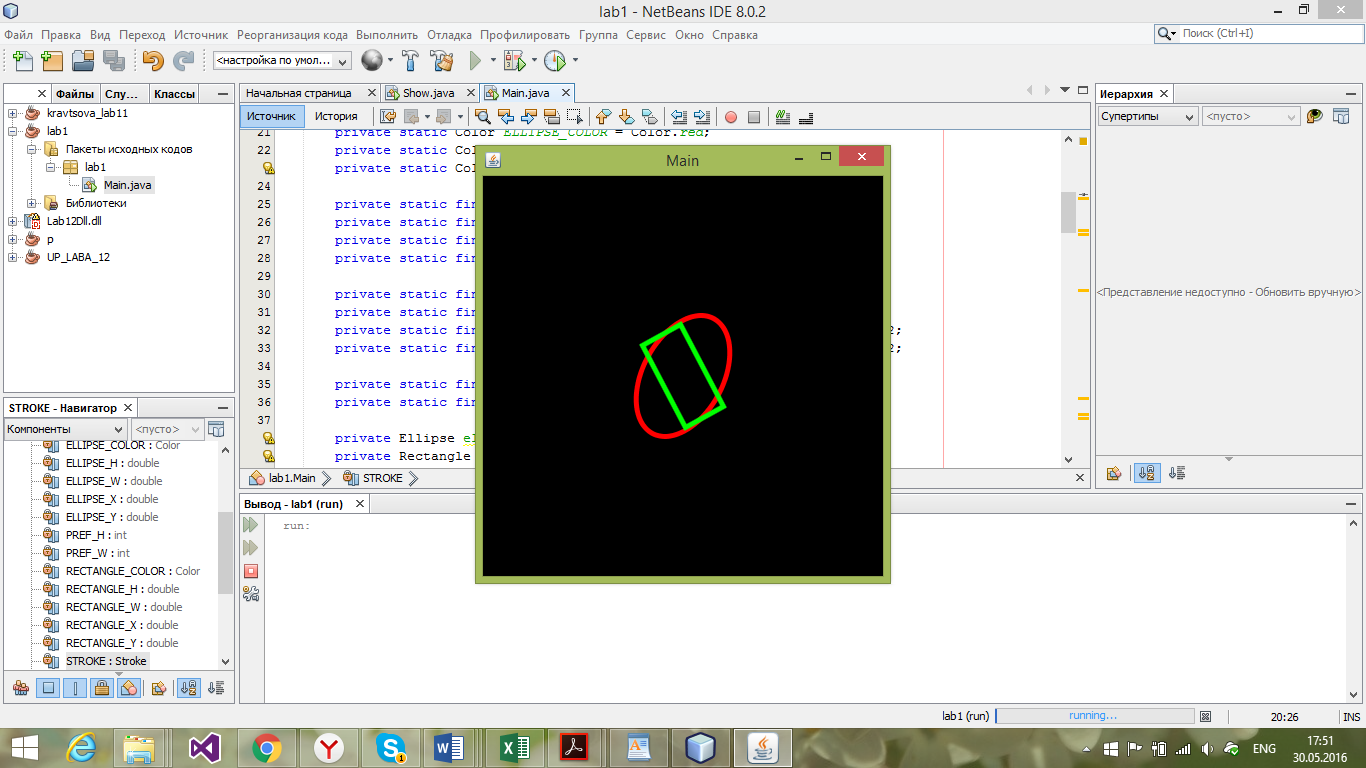
Для изображения указанной в задании фигуры создать класс, реализующий интерфейс Shape. Выполнить перемещения указанной в задании фигуры с помощью аффинного преобразования координат. Выполнить рисунок в окне апплета или фрейма с выбранной толщиной границы фигуры, цветом границы и цветом внутренней области (вводить толщину и цвет в качестве аргументов ваших программ или параметров апплета).

* 1. ***Решение задачи:***

Для создания прямоугольника был создан класс, который наследуется от класса java.awt.geom.Зусефтпду2D.Double, а для круга – от класса java.awt.geom.Ellipse2D.Double. Таким образом оба класса реализуют интерфейс Shape.

Вращение фигуры задается при помощи класса java.awt.geom.AffineTransform.

* 1. ***Результат:***

 Рисунок 1.1 – Результат работы приложения

1. **Выводы**

Класс Graphics2D, в отличие от своего предка Graphics, позволяет использовать вещественные координаты.

Для удобного способа задания параметров отрисовки контуров можно использовать класс BasicStroke, что реализует интерфейс Stroke.

Для создания своего собственного класса, который представляет собой фигуру, достаточно реализовать интерфейс Shape. При правильной реализации, Graphics2D сможет корректно отобразить нашу фигуру.

В пакете java.awt.geom содержится набор уже написанных классов фигур, на основании которых с небольшими усилиями можно построить свои простые фигуры.

Аффинные преобразования при использовании графики в Java не обязательно делать вручную, а можно использовать класс AffineTransform.

**ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №2**

***Цель:***

Научиться методам обработки изображений в Java 2D API, к которым относятся сглаживание, применение фильтров, использование внеэкранных буферов, текстурные и градиентные заливки, работа с альфа-каналами.

***Предметная область:***

Двумерная графика.

1. **Краткие теоретические сведения**

Для обработки изображений в Java 2D API введены следующие классы и интерфейсы:

* Интерфейс Paint, описывающий заливку фигур. Этот интерфейс реализуют классы Color, что определяет сплошную заливку, GradientPaint, описывающий градиентную заливку, и TexturePaint, который заполняет фигуру текстурой.
* Интерфейс Composite, описывающий композиции цветов. AlphaComposite является одним из классов, реализующих этот интерфейс. Он комбинирует цвета, основываясь на их прозрачности (значении альфа-канала).
* Класс RenderingHints для настройки параметров отрисовки как, например, сглаживание.
* Класс BufferedImage, для работы с внеэкранными буферами.
* Интерфейс BufferedImageOp, при помощи которого определяются фильтры обработки изображений

1. **Задание**
   1. ***Постановка задачи:***

Изобразить дорожный знак: вопросительный знак в квадрате, цвет квадрата и символа – желтый, цвет фона – серый с градиентной заливкой справа-налево. Фильтра: Image negative

Для изображения указанной в задании фигуры создать класс, реализующий интерфейс Shape. Создать указанный фильтр изображения. При тестировании вывести фигуру без фильтра и с фильтром. Смоделировать освещение и тень от объекта при помощи альфа-канала и/или механизма обработки изображения. При рисовании использовать сглаживание, внеэкранный буфер и преобразования координат.

* 1. ***Решение задачи:***

Для создания знака был создан класс, который наследуется от класса java.awt.geom.Rectangle2D.Double. Для отрисовки знака был создан метод draw(Graphics2D g), который рисует знак в соответствующую графику. Размер текста внутри подгоняется под размер знака.

Тень от знака рисуется при помощи аффинных преобразований над исходным знаком. Тень полупрозрачна. Для градиентной заливки используется класс GradientPaint.

При отрисовке используются два буфера: один для исходного изображения знака, один для изображения после применения фильтра.

Изображения отрисовываются в компоненте DrawComponent. Сам компонент располагается на JFrame.

* 1. ***Результат:***

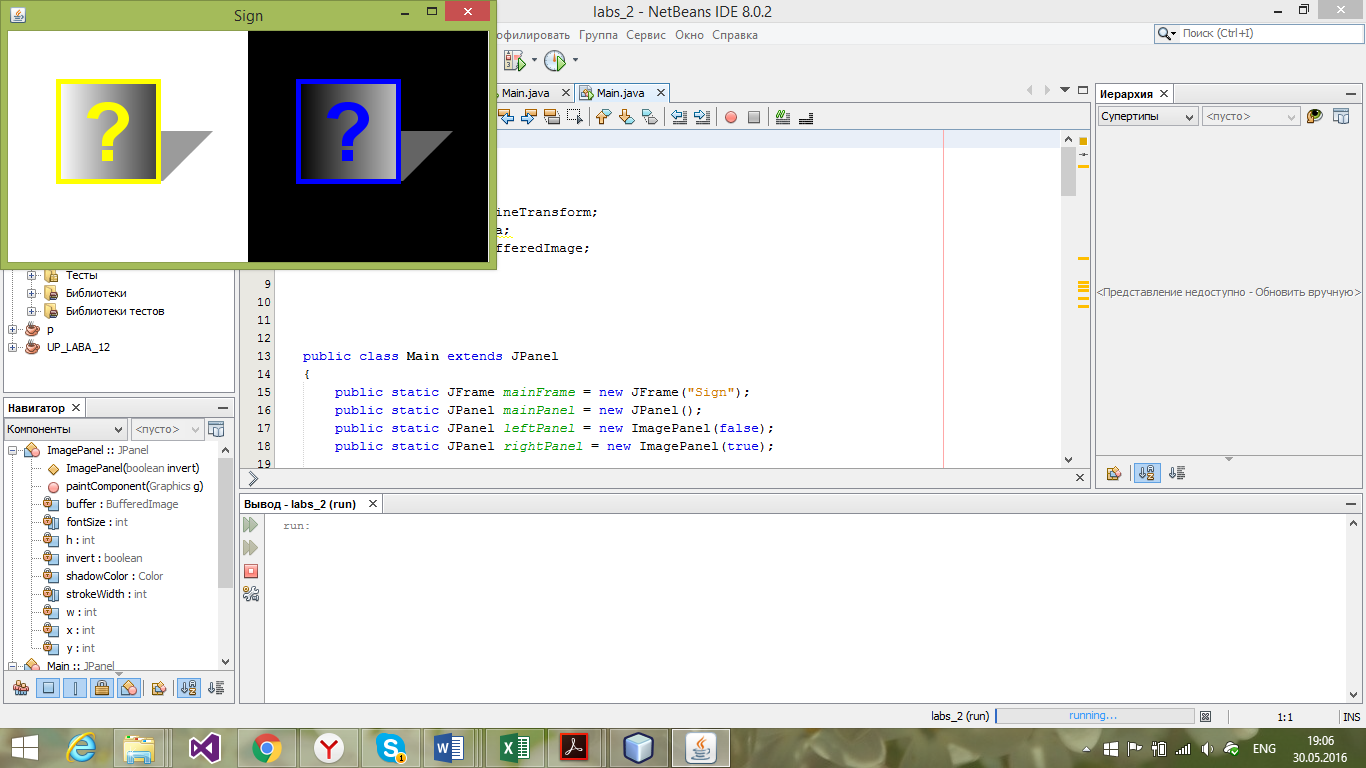


Рисунок 2.1 – Результат работы приложения

1. **Выводы**

Работа с внеэкранными буферами незначительно отличается от работы напрямую. Необходимо только воспользоваться методом BufferedImage.getGraphics(), после чего работать с привычным классом Graphics.

Для установки сглаживания необходимо установить ключи ANTIALIASING и TEXT\_ANTIALIASING в объекте класса RenderHints.

Множество фильтров обработки изображений уже реализованы в Java и они достаточно просты в использовании.

**ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №3**

***Цель:***

Научиться создавать свои произвольные фигуры и контуры с использованием интерфейсов Shape и Stroke из Java 2D API.

***Предметная область:***

Двумерная графика.

1. **Краткие теоретические сведения**

Для создания своего пользовательского контура необходимо создать класс, реализующий интерфейс Stroke. Этот интерфейс содержит только метод createStrokedShape. Он принимает исходную фигуру в качестве параметра и возвращает уже новую фигуру. То, как будет выглядеть эта фигура, будет зависеть от реализации.

Для создания своей пользовательской фигур необходимо создать класс, реализующий интерфейс Shape. Основных методом этого интерфейса является метод getPathIterator, который возвращает итератор, определяющий координаты точек фигуры, ее вид и т.д. Также в интерфейсе содержатся дополнительные методы:

* Методы contains, которые определяют, содержит ли фигуpa точку или прямоугольник.
* Методы intersects, которые определяют, пересекается ли какая-то часть фигуры с заданным прямоугольником.

1. **Задание**
   1. ***Постановка задачи:***

Разработать пользовательский класс Shape реализующий рисование указанной алгебраической линии. Разработать пользовательский класс Stroke для отображения указанного контура. Создать приложение или апплет для тестирования и демонстрации разработанных классов.

Линия: Трёхлепестковая роза

(x2 + y2)2 = a(3x2y –y3)

* 1. ***Решение задачи:***

Для представления линии используется класс Line, реализующий интерфейс Shape. При подстановке поля класса x в формулу функция возвращает определенный результат, который будет соответствовать координате y. Также имеются два поля, ограничивающие область задания функции. Метод getPathIterator() возращает объект внутреннего класса LineIterator, который содержит методы, необходимые для задания сегментов для отрисовки.

Контур линии задается классом LineStroke, реализующим интерфейс Stroke. Рисунок выполняется на типичном апплете методом draw объекта класса Graphics2D

* 1. ***Результат:***

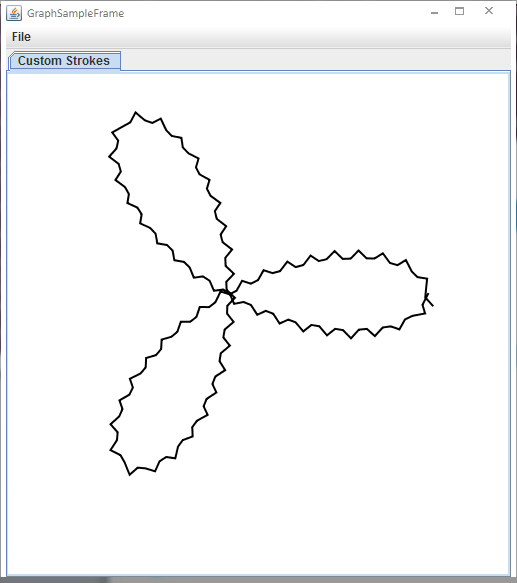


Рисунок 3.1 – Результат работы приложения

1. **Выводы**

Основной задачей при создании своей фигуры является созданием для нее PathIterator’а. Именно он отвечает за вид фигуры.

При создании своей фигуры можно рисовать не только прямыми линиями, но также и кривыми Безье (SEG\_CUBICTO, SEG\_QUADTO).

Процесс создания своей кисти не сильно отличается от создания PathIterator’а для фигуры, за исключением того, что нужно ориентироваться на уже существующую фигуру.

**ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №4**

***Цель:***

Научиться пользоваться системой печати с использованием интерфейсов Printable и Pageable из Java API.

***Предметная область:***

Двумерная графика, редактирование текста.

1. **Краткие теоретические сведения**

Для выполнения печати в Java просто нужно получить объект Graphics, который использует в качестве поверхности рисования принтер. Если есть объект Graphics, то можно печатать текст и выводить графику на принтер, как будто это обычный экран.

Единственная хитрость относится к получению объекта Graphics, связанного с принтером. Для этой задачи используют класс PrinterJob. Он позволяет распечатать объект, реализующий интерфейс Printable. Также с его помощью можно вывести диалог выбора принтера и параметров печати, которые также можно задавать вручную при помощи объекта класса HashPrintRequestAttributeSet, в который помещаются соответствующие атрибуты.

При печати в Java также используется интерфейс Pageable, которые представляет собой интерфейс работы с многостраничным документом.

1. **Задание**
   1. ***Постановка задачи:***

Модифицируйте вашу программу следующим образом. В демонстрационное приложение

добавьте возможность печати небольшого отчёта о решении задания №3. Отчёт должен

содержать следующее:

- рисунок с подписью алгебраической линии вашего задания

- исходный текст класса Shape, реализующий рисование указанной алгебраической линии.

При печати используйте режим альбомной ориентации страницы и двустороннюю печать.

Рисунок должен занимать не более половины страницы, при печати выровнять его по

горизонтали.

* 1. ***Решение задачи:***

Для представления линии используется класс Line, реализующий интерфейс Shape. Циссоиду можно представить в виде двух половин, которые представлены отдельными функциями. При подстановке поля класса x в формулу функция возвращает определенный

результат, который будет соответствовать координате y. Также имеются два поля, ограничивающие область задания функции. Метод getPathIterator() возращает объект внутреннего класса LineIterator, который содержит методы, необходимые для задания сегментов для отрисовки.

Контур линии задается классом LineStroke, реализующим интерфейс Stroke.

Рисунок выполняется на типичном апплете методом draw объекта класса Graphics2D.

На этот апплет были добавлены кнопки печати отчета и выхода. При нажатии на кнопку печати обрабатывается событие и вызывается метод print() объекта класса PrinterJob. Как сказано выше, этот класс отвечает за диалог с пользователем при настройке печати.

Непосредственно за печать отвечает класс PrintableDoc, реализующий интерфейс Printable. В нем вычисляются необходимые для печати задачи: количество страниц отчета, количество строк на странице, размеры строк, координаты рисунка и т.п.

* 1. ***Результат:***

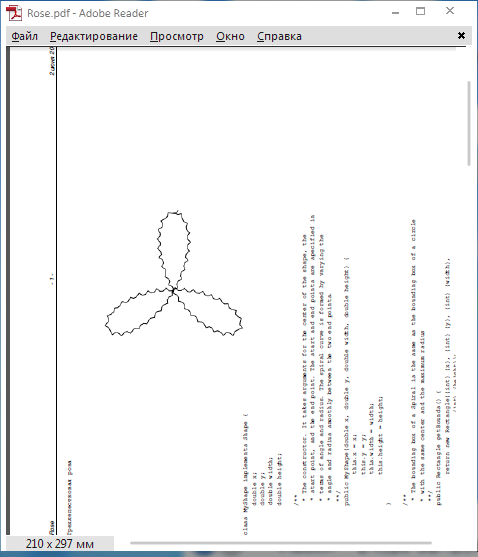


Рисунок 4.1 – Скриншот приложения, отображающего данные, отправленные на печать

1. **Выводы**

Работа с принтерами обеспечивается классом PrinterJob, в котором уже есть готовый графический интерфейс для настройки печати.

Для того, чтобы узнать, сколько текст будет занимать места на странице можно использовать класс FontMetrics.

Если текст не помещается на одну страницу, то разбивку на страницы необходимо делать самостоятельно.

**ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №5**

***Цель:***

Научиться обмену объектами в компонентах Swing средствами пакетов *java.awt.datatransfer* и *java.awt.dnd* в виде drag-and-drop.

***Предметная область:***

Двумерная графика, разработка графических интерфейсов.

1. **Краткие теоретические сведения**

Пакет *java.awt.datatransfer* предоставляет возможность передачи данных между приложениями и поддерживает метод обмена данными типа «вырезание и вставка» (cut-and-paste).

Пакет *java.awt.dnd* поддерживает метод передачи данных типа «перетаскивание» (drag-and-drop).

Класс *java.awt.datatransfer.DataFlavor* является центральным в процессе передачи данных; он представляет тип данных, подлежащих передаче. Каждый формат данных (data flavor) содержит удобочитаемое имя, объект *Class*, указывающий тип передаваемых данных, и тип MIME, определяющий кодировку, используемую при передаче данных.

В классе *DataFlavor* предопределена пара наиболее часто используемых форматов для передачи строк и списков объектов *File*. Кроме этого, в нем определено несколько типов MIME, используемых этими форматами.

Интерфейс *java.awt.datatransfer.Transferable* является важной частью механизма передачи данных. Этот интерфейс задает три метода, которые должны быть реализованы каждым объектом, желающим сделать свои данные доступными для передачи:

* *getTransferDataFlavor()* – возвращает массив всех типов *DataFlavor*, которые он может использовать для передачи своих данных;
* *isDataFlavorSupported()* – проверяет, поддерживает ли объект *Transferable* данный формат
* *getTransferData()* – возвращает данные в формате, соответствующем запрошенному *DataFlavor*.

Архитектура передачи данных основывается на механизме сериализации объектов как на одном из средств передачи данных между приложениями.

Начиная с Java 1.2, была введена поддержка обмена данными методом «перетаскивание» (drag-and-drop). Программный интерфейс для этого механизма находится в пакете *java.awt.dnd* и основан на той же архитектуре *DataFlavor* и *Transferable*, что и механизм вырезания и вставки.

1. **Задание**
   1. ***Постановка задачи:***

Для выполнения задания используется ваш вариант решения задания №3. Модифицируйте вашу программу следующим образом.

Создайте тестовое приложение, добавьте в ваш класс рисования алгебраической линии возможность «перетаскивание» (drag-and-drop). Реализуйте необходимые интерфейсы в классе и в приложении для демонстрации «перетаскивания» алгебраической линии между несколькими копиями тестового приложения.

При реализации интерфейса тестового приложения следуйте рекомендациям стандарта CUI (Common User Interface).

* 1. ***Решение задачи:***

В режиме рисования в месте клика рисуется фигура из задания 3, а в режиме перемещения фигура перемещается в другое окно. Она сериализуется в первом приложении и дересеализуется во втором приложении.

* 1. ***Результат:***

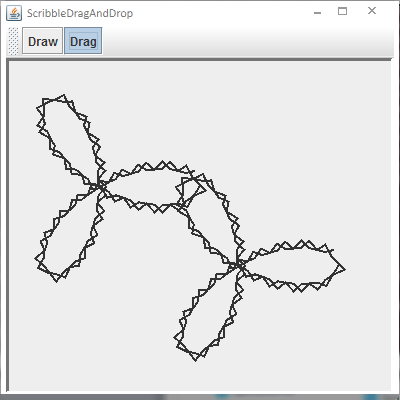


Рисунок 5.1 – Результат работы приложения

1. **Выводы**

Для того, чтобы наш созданный класс можно было использовать в передаче, необходимо, чтобы он реализовывал интерфейс *Transferable*. Его реализация значительно упрощается, если мы можем использовать механизм сериализации через реализацию интерфейса *Serializable*.

Основная задача в реализации механизма drag-and-drop состоит в реализации интерфейсов *DragGestureListener*, *DragSourceListener* и *DropTargetListener*.

Drag-and-drop может работать как в пределах одного приложения, так и между несколькими.

**ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №6**

***Цель:***

Научиться использовать компоненты Swing *javax.swing.JTree* и *javax.swing.JTable*.

***Предметная область:***

Научные конференции.

1. **Краткие теоретические сведения**

Класс *JTable* отображает табличные данные. Он особенно легко применяется, когда данные организованы в виде массива массивов. Если же это не так, то необходимо создать модель для таблицы в виде класса, реализующего интерфейс *javax.swing.table.TableModel*. Эта модель будет служить переводчиком между данными и компонентом *JTable*.

Компонент *JTree* применяется для отображения данных, имеющих структуру дерева. Если данные имеют форму вложенных массивов, векторов или хеш-таблиц, можно передать корневой узел структуры данных конструктору *JTrее*, и он их отобразит. Данные, имеющие древовидную структуру, обычно имеют иную форму, чем перечисленные выше. Отобразить такие данные можно, реализовав интерфейс *javax.swing.Tree.TreeModel*, чтобы проинтерпретировать данные способом, пригодным для использования компонентом *JTree*.

1. **Задание**
   1. ***Постановка задачи:***

Разработать систему классов/интерфейсов для расписания научных конференций. Данные необходимо упорядочить по атрибутам/свойствам товаров, предметов и т.п. в виде дерева.

Разработать графическое приложение для ввода/отображения данных расписания автовокзала. При отображении структуры данных в виде дерева реализовать интерфейс *javax.swing.Tree.TreeModel*. Листья дерева отображать в виде таблицы, реализуя интерфейс *javax.swing.table.TableModel*.

При реализации интерфейса следовать рекомендациям стандарта CUI (Common User Interface).

* 1. ***Решение задачи:***

Данные о каждой статье хранятся в объектах класса *Article*. Класс MyTableModel является наследником класса AbstractTableModel. В нем хранится массив объектов Article.

На главной форме находятся дерево, при помощи которого можно осуществлять вывод статей, и таблица, в которой отображаются записи. В дерево можно добавить узел (объект класса MyTreeNode). Этот объект так же содержит массив объектов Article. При двойном нажатии на этот уел дерева данные из объекта узла отправляются в модель таблицы, и после обновления таблицы выводятся на экран. Так же можно в узел добавить статью. Создается новый объект Article и добавляется в массив к узлу дерева, в который добавляем.

* 1. ***Результат:***

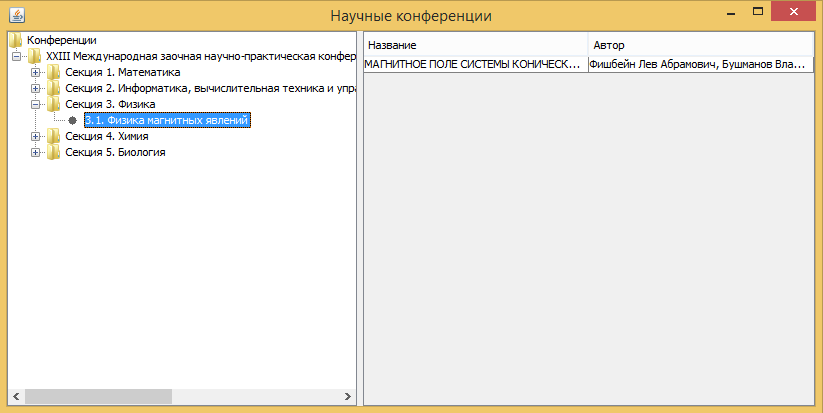


Рисунок 6.1 – Результат работы приложения

1. **Выводы**

Для того чтобы *JTree* или *JTable* отрисовали изменения в таблице, необходимо в модели сохранять слушателей и во время изменения данных их оповещать. При установке модели, компоненты сами подписываются в ней.

Сортировку в таблице можно не реализовывать самому, а использовать автоматически сгенерированную, указав *setAutoCreateRowSorter(true)*.

Дерево и таблица могут использовать один и тот же объект как модель, если класс этой модели реализует оба интерфейса *TableMo7del* и *TreeModel*.

**ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №7**

***Цель:***

Научиться разработке своих собственных компонентов *JavaBeans*.

***Предметная область:***

Графический интерфейс пользователя.

1. **Краткие теоретические сведения**

Программный интерфейс *JavaBeans* предоставляет среду для разработки многократно используемых, встраиваемых, модульных программных компонентов. Спецификация *JavaBeans* дает следующее определение компонента (bеаn): «многократно используемый программный компонент, которым можно манипулировать в визуальных средах разработки». Это достаточно свободное определение; компоненты могут принимать самые разные формы. На самом простом уровне компонентами *JavaBeans* являются все отдельные графические компоненты, тогда как на гораздо более высоком уровне в качестве компонента может также функционировать встраиваемое графическое приложение. Однако большинство компонентов находятся где-то между этими двумя крайностями.

Одной из целей модели *JavaBeans* является обеспечение взаимодействия с аналогичными компонентными средами. Так, например, обычная Windows программа может с помощью соответствующего моста или компонента-обертки пользоваться компонентом Java так, как будто бы он является компонентом СОМ или ActiveX.

Любой объект, удовлетворяющий определенным базовым правилам и соглашениям об именах, может считаться компонентом *JavaBeans*.

Класс должен:

* Класс должен иметь конструктор без параметров, с модификатором доступа *public*. Такой конструктор позволяет инструментам создать объект без дополнительных сложностей с параметрами.
* Свойства класса должны быть доступны через *get*, *set* и другие методы (так называемые методы доступа), которые должны подчиняться стандартному соглашению об именах. Это легко позволяет инструментам автоматически определять и обновлять содержание bean’ов. Многие инструменты даже имеют специализированные редакторы для различных типов свойств.
* Класс должен быть [сериализуем](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B8%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F#Java). Это даёт возможность надёжно сохранять, хранить и восстанавливать состояние bean независимым от платформы и виртуальной машины способом.

1. **Задание**
   1. ***Постановка задачи:***

Разработать простой компонент на базе класса *Canvas* – круглая заливка. Создать файл манифеста и упаковать компонент вместе с исходным кодом разработанных классов. При разработке поместить все классы в пакет *bsu.fpmi.educational\_practice2016*.

Создать тестовое приложение в *NetBeans* с использованием разработанного компонента.

* 1. ***Решение задачи:***

В качестве компонента *JavaBeans* выступает класс *Circle*. В нем присутствуют свойства *get/setLineDiameter* и *get/setLineColor*, которые отвечают за диаметр и цвет заливки соответственно. Класс унаследован от класса *Canvas*.

Собранный \*.jar файл был добавлен как компонент в проект.

* 1. ***Результат:***

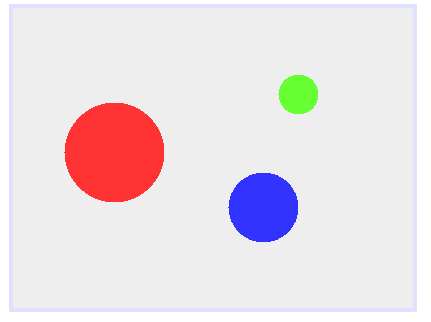


Рисунок 7.1 – Результат работы приложения

1. **Выводы**

Для того, чтобы класс соответствовал спецификации *JavaBeans*, он просто должен реализовывать интерфейс *Serializable*, иметь публичный конструктор по умолчанию и для свойств иметь соответствующие методы *get* и *set*.

Реализация компонента как *JavaBeans* компонента позволяет его легко использовать в визуальных редакторах интерфейсов, например, в *NetBeans*.

**ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №8**

***Цель:***

Научиться разработке своих собственных компонентов *JavaBeans* с собственными событиями и свойствами.

***Предметная область:***

Графический интерфейс пользователя.

1. **Краткие теоретические сведения**

Cхема работы модели событий:

Компонент Java определяет событие, если он предоставляет методы *add* и *remove* для регистрации (добавления) и удаления объектов-слушателей данного события.

Приложение, которое хочет получать уведомление о том, что произошло событие определенного типа, использует эти методы для регистрации объекта-слушателя событий соответствующего типа.

Когда происходит событие, компонент Java уведомляет зарегистрированные приемники путем передачи событийного объекта, который описывает событие, методу, определенному интерфейсом слушателя событий.

Свойство *–* это часть внутреннего состояния компонента Java. Его можно установить программно и/или получить его значение – обычно при помощи стандартной пары методов доступа *get* и *set*.

В дополнение к обычным свойствам *JavaBeans API* поддерживает несколько специализированных подтипов.

Индексированное свойство – это свойство, значением которого является массив, а также методы доступа, которые позволяют обращаться как к отдельным элементам массива, так и ко всему массиву в целом.

Связанное свойство – это свойство, которое посылает событие *PropertyChangeEvent* любым заинтересованным объектам *PropertyChangeListener*, когда значение свойства изменяется.

Ограниченное свойство – это свойство, любые изменения в котором могут быть заблокированы любым заинтересованным слушателем. Когда меняется значение ограниченного свойства компонента Java, он должен выслать *PropertyChangeEvent* списку заинтересованных объектов *VetoableChangeListener*. Если любой из этих объектов вызывает исключение *PropertyVetoException*, то значение свойства не меняется, а исключение *PropertyVetoException* возвращается методу, который устанавливает свойство.

1. **Задание**
   1. ***Постановка задачи:***

Разработать компонент: однострочный статический текст и кнопка; свойствами являются текст, текст кнопки, символ подтверждения. События генерируются при нажатии на кнопку или вводе символа. Создать файл манифеста и упаковать компонент. При разработке поместить все классы в пакет *bsu.fpmi.educational\_practice2016*. Компонент должен реализовать класс *BeanInfo* с информацией о компоненте. Создать тестовое приложение в *NetBeans* с использованием разработанного компонента.

Создать интерфейсные компоненты с реализацией собственного события *AcсeptEvent*. Определить интерфейс слушателя *AcсeptListener*. Передавать слушателю события информацию о том, в результате чего произошло событие. Событие генерируется при нажатии на кнопку.

* 1. ***Решение задачи:***

В качестве компонента *JavaBeans* выступает класс *ButtonAndText*. В нем присутствуют свойства *get/setButtonText* для установки текста кнопки и *get/setLabelText* для установки текста надписи. Класс унаследован от класса *JPanel*.

Компонент включает в себя событие *AcceptEvent*. Для добавления и удаления подписчиков реализованы методы *add/removeAcceptListener*. Событие возбуждается, когда происходит нажатие на кнопку. Событие распознается средой *NetBeans*.

Для того чтобы было описание создано разработанных свойств и событий, был создан класс *ButtonAndTextBeanInfo*. Большая его часть была автоматически сгенерирована средой *NetBeans*.

* 1. ***Результат:***

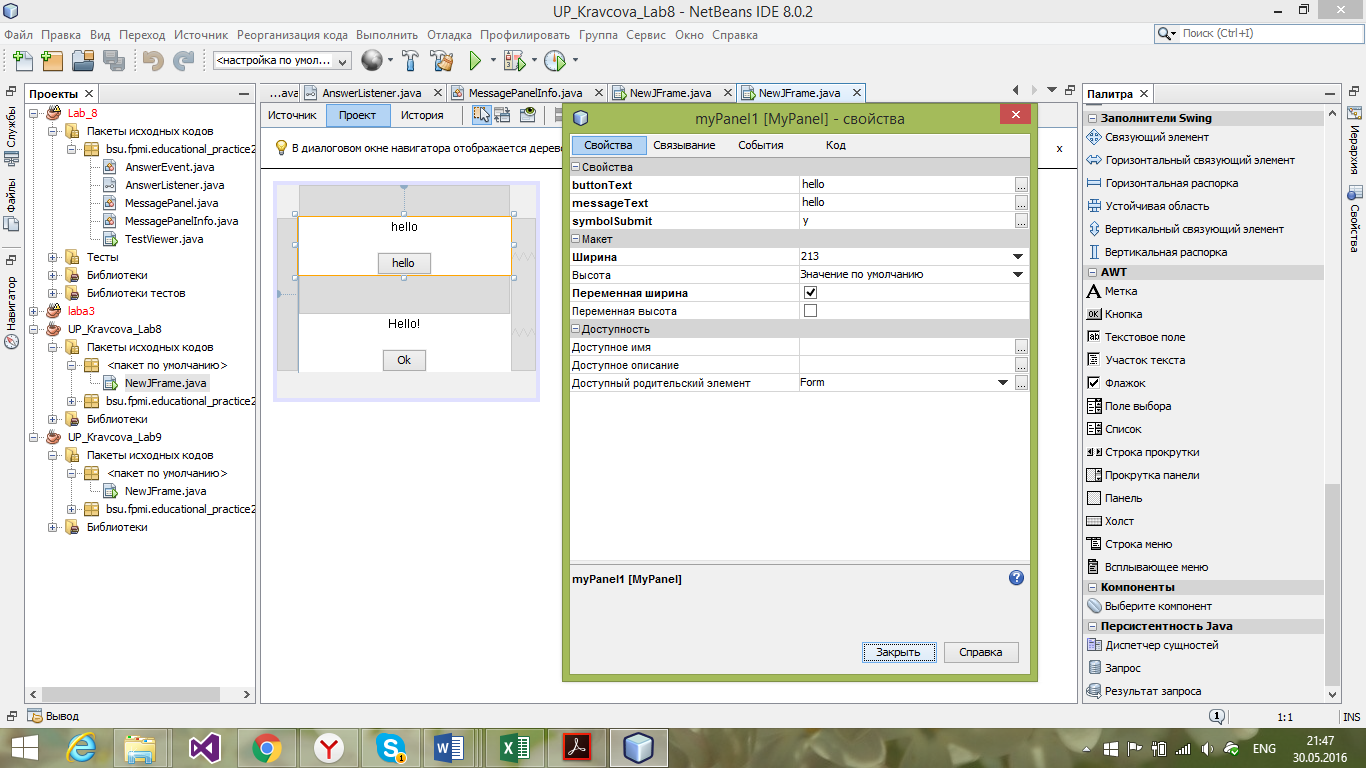


Рисунок 8.1 – Использование компонента в *NetBeans*

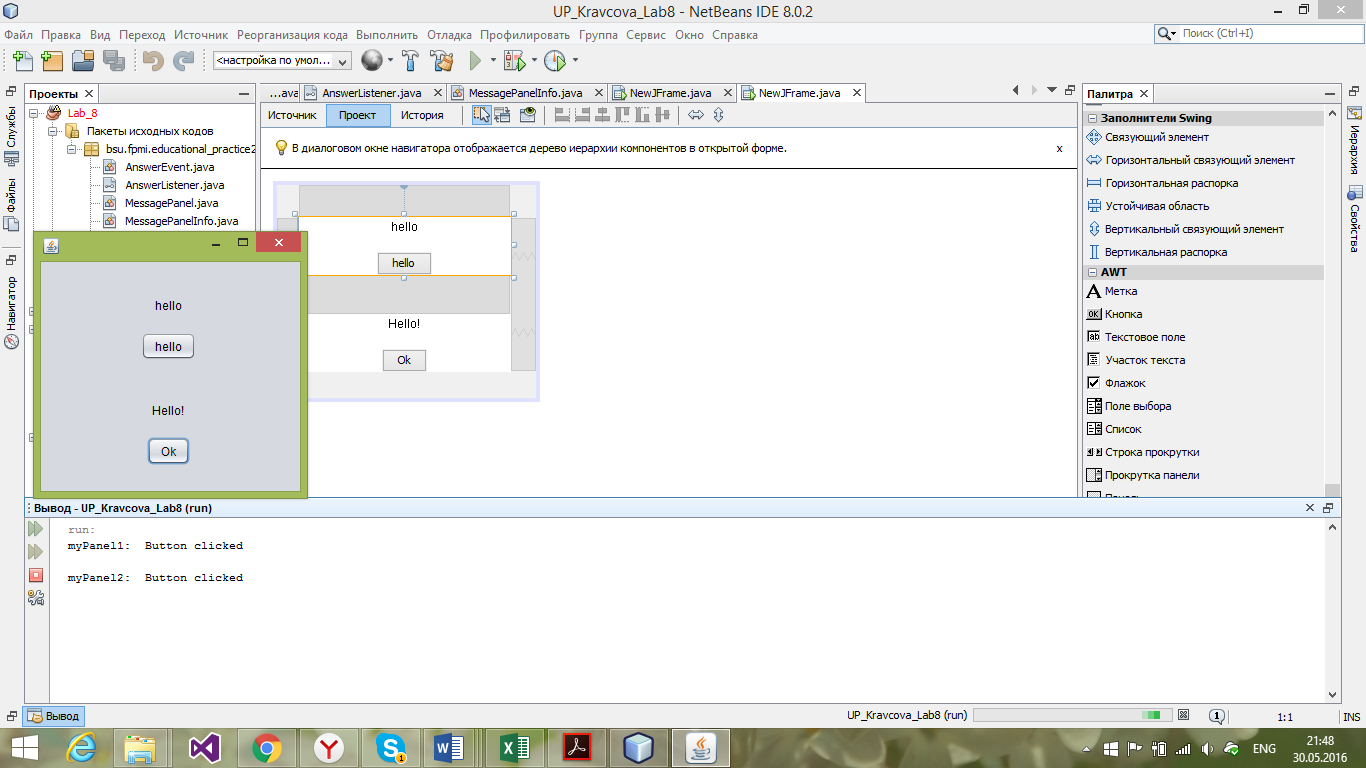


Рисунок 8.2 – Результат работы приложения

1. **Выводы**

Для того чтобы созданное событие было распознано в IDE, разработанный слушатель должен наследоваться от класса *EventListener*, а само событие должно наследоваться от *EventObject*.

Для работы с событиями должны быть реализованы *add* и *remove* методы в компоненте.

Описания событий и свойств можно определить в классе, реализующем интерфейс *BeanInfo*.

**ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №9**

***Цель:***

Научиться разработке своих собственных редакторов свойств для компонентов *JavaBeans*.

***Предметная область:***

Графический интерфейс пользователя.

1. **Краткие теоретические сведения**

Компонент может предоставлять используемые вспомогательные классы *PropertyEditor*. *PropertyEditor* является достаточно гибким интерфейсом, предоставляющим компоненту возможность сообщить контейнеру, как нужно отображать и редактировать значения свойств определенных типов.

Для свойств обычных типов, таких как строки, числа, шрифты и цвета, контейнер всегда предоставляет простые редакторы свойств. Однако если у вашего компонента есть свойство нестандартного типа, вы должны зарегистрировать для этого типа свой редактор свойств. Самый простой способ зарегистрировать редактор свойств состоит в регистрации редактора свойств путем вызова метода *PropertyEditorManager.registerEditor()*, например, из конструктора класса *BeanInfo*. В случае если вы будете вызывать этот метод из самого компонента, компонент будет зависеть от класса редактора свойств, поэтому редактор будет привязан к компоненту при использовании того в приложениях, что является нежелательным. Другой способ регистрации редактора свойства состоит в использовании в классе *BeanInfo* объекта *PropertyDescriptor* с целью указания *PropertyEditor* для отдельного свойства.

1. **Задание**
   1. ***Постановка задачи:***

Создать собственный редактор для каждого свойства компонента из предыдущей лабораторной работы. Каждый редактор должен ограничивать возможные значения свойства, предоставляя выбор из списка трёх – пяти допустимых значений. Зарегистрировать редакторы в классе BeanInfo компонента.

Создать настройщик компонента, который позволит изменять списки допустимых значений для свойств созданного компонента.

* 1. ***Решение задачи:***

Для свойств *labelText* и *buttonText* были созданы редакторы *LabelTextPropertyEditor* и *ButtonTextPropertyEditor* соответственно. Оба они наследуются от класса *StringPropertyEditor*, в котором реализован весь функционал. В методе *getJavaInitializationString* строки названий экранируются, чтобы нормально поддерживать названия с символами кавычек и слешей.

Для редактирования списка допустимых значений используется класс *EditorComponent*. Компонент представляет собой панель с редактируемым *JComboBox* и с кнопкой добавления. Для добавления нового значения, необходимо его ввести в бокс и нажать кнопку *Add*.

* 1. ***Результат:***

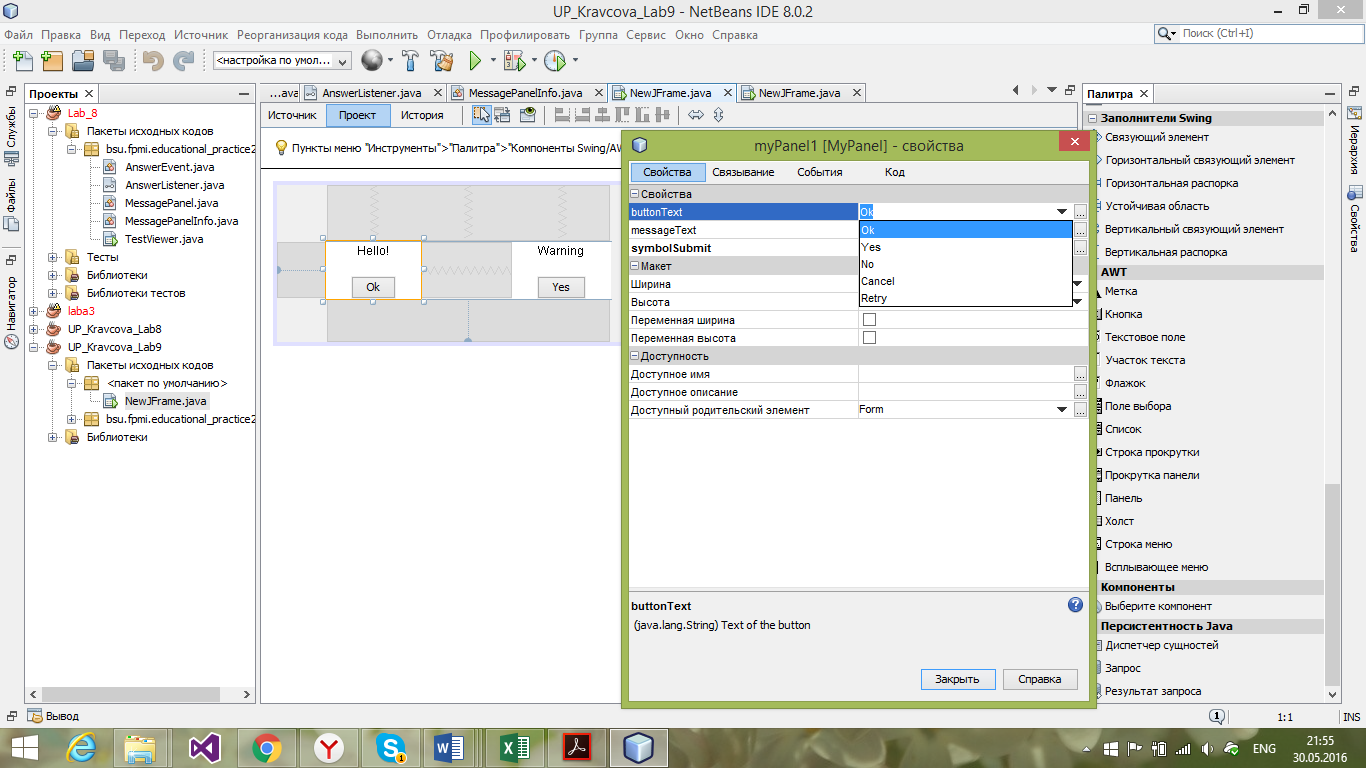


Рисунок 9.1 – Использование компонента в *NetBeans* c выбором допустимых значений

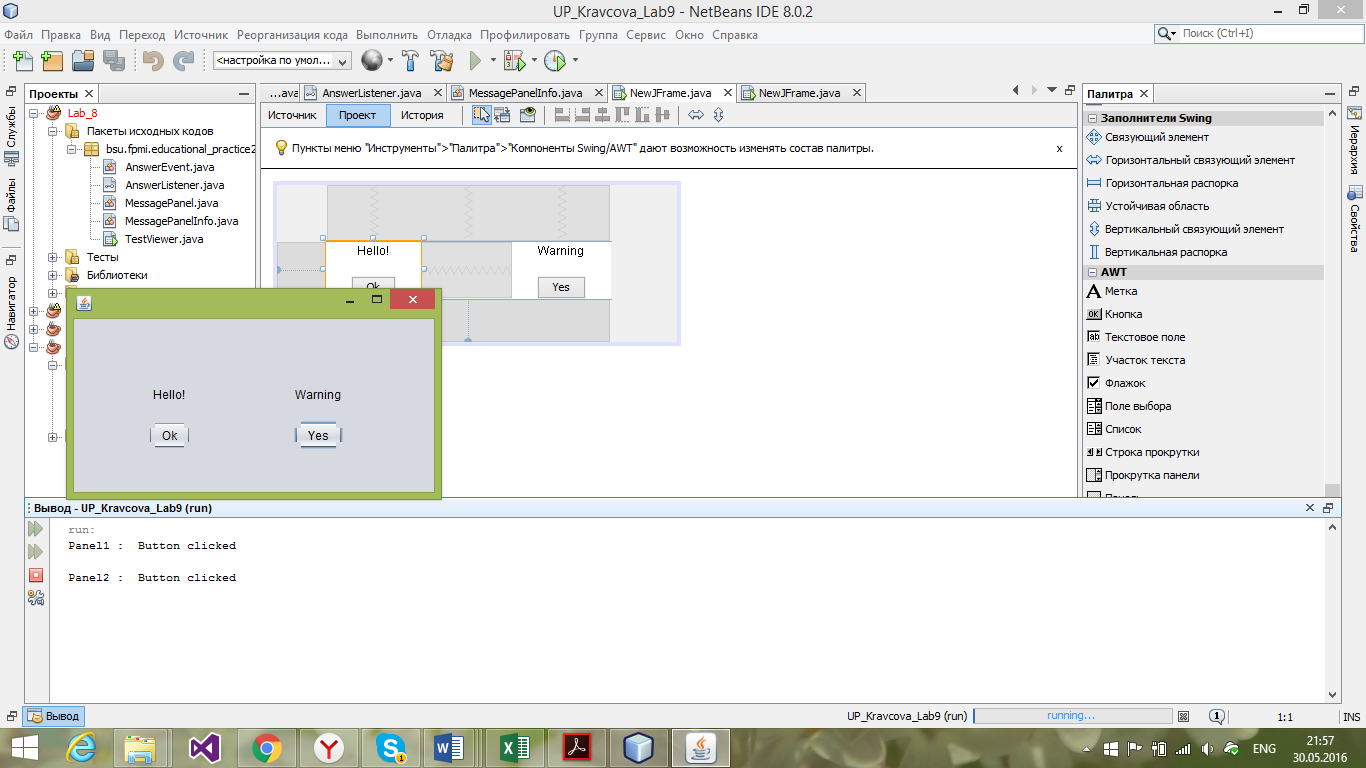


Рисунок 9.2 – Результат работы приложения

1. **Выводы**

Настройку свойств собственных компонентов можно контролировать, если для каждого свойства создать свой класс, реализующий *PropertyEditor*. Он будет отвечать за редактор настроек привязанного к нему свойства.

Для настройки своих свойств можно создать свой класс-наследник *Component*, в котором сделать удобный для редактирования GUI.

**ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №10**

***Цель:***

Научиться обработке текстов и работе с текстовыми файлами. Научиться разработке графических интерфейсов.

***Предметная область:***

Графический интерфейс пользователя. Обработка текста.

1. **Краткие теоретические сведения**

Создание GUI на основе Java происходит в четыре этапа:

***1. Создание и конфигурирование компонентов***

Компонент GUI создается путем вызова его конструктора. Создав компонент, можно сконфигурировать его, задав одно или несколько его свойств. Следует познакомиться с документацией на используемый компонент, чтобы узнать, каких аргументов ожидает конструктор и какие методы можно применять для конфигурирования.

***2. Помещение компонента в контейнер***

Все компоненты должны размещаться в контейнере. Все контейнеры в Java являются подклассами *java.awt.Container*. Контейнер – это вид компонента, поэтому контейнеры могут быть помещены и часто помещаются в другие контейнеры. Чтобы поместить компонент в контейнер, его просто передают методу *add()* контейнера.

***3. Размещение или компоновка компонентов***

Необходимо задать положение и размер каждого компонента в его контейнере так, чтобы GUI хорошо смотрелся. Хотя можно включить положение и размер каждого компонента в код программы, гораздо чаще применяют объект *LayoutManager* для автоматического размещения компонентов в контейнере в соответствии с определенными правилами компоновки, задаваемыми конкретным, выбранным программистом объектом *LayoutManager*.

***4. Обработка событий, генерируемых компонентами***

При взаимодействии пользователя с компонентами GUI при помощи клавиатуры или мыши эти компоненты генерируют, или активизируют, события. Событие – это просто объект, содержащий информацию о действиях пользователя. Последним шагом при создании GUI является добавление слушателей событий – объектов, которым посылаются уведомления о том, что произошло событие, и которые адекватно отвечают на это событие. Например, кнопке Quit нужен слушатель, который вызовет выход из приложения.

1. **Задание**
   1. ***Постановка задачи:***

* Разработайте приложение для выполнения несложных действий с текстовым файлом.
* При реализации интерфейса следуйте рекомендациям стандарта CUI (Common User Interface). Главное меню приложения обязательно должно содержать следующие команды (в формате Подменю/Команда):

*File/Quit* – завершение приложения после подтверждения пользователя

*View/Font* – выбор шрифта для компонентов интерфейса

*View/ Look and Feel* – выбор стиля интерфейса (как в примерах)

*Help/About* – вывод диалогового окна которое печатает условие задачи и

информацию о разработчике (ФИО студента, группа, курс, факультет, ВУЗ).

*(команды)* – команды необходимые для решения вашего варианта задачи,

например, File/Open для выбора файла и т.п.

* Приложение должно корректно обрабатывать команды системного меню (изменение размеров окна, команды «Закрыть», «Свернуть» и т.д.).

Программа читает текст из выбранного файла, разбивает его на предложения. В клиентской области создать две панели, разделённые по вертикали, в левой части отобразить содержимое файла, в правой части отобразить все предложения, переводя все слова восклицательных предложений в верхний регистр, для каждого предложения использовать отдельный компонент JLabel.

* 1. ***Решение задачи:***

Графический интерфейс приложения был создан при помощи редактора форм в IDE *Netbeans*.

Созданы пункты меню *File/Open* (открывает текстовый файл и начинает обработку предложений), *File/Quit* (выход из приложения), *View/Font* (выбор отображаемого шрифта), *View/Look and Feel* (настройка *L&F*) и *Help/About*. При наведении на них снизу появляется подсказка.

Для разбивки текста на предложения и предложений на слова использовались объекты класса *BreakIterator*.

* 1. ***Результат:***

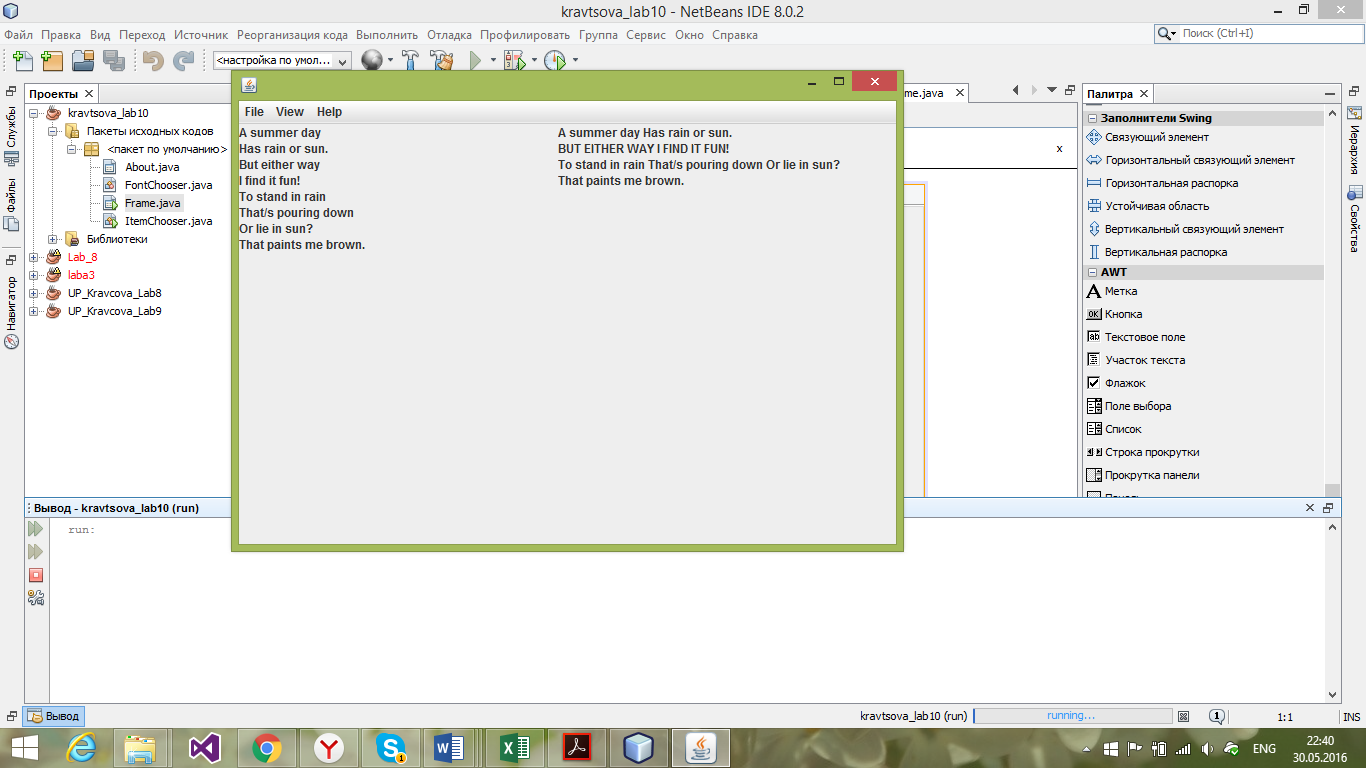


Рисунок 10.1 – Результат работы приложения

1. **Выводы**

Для разбивки текста на строки, предложения, слова и т.д. с учетом настроек локали можно использовать класс *BreakIterator*.

Для быстрой разработки интерфейсов можно использовать визуальный редактор форм в IDE *NetBeans*, который генерируем соответствующий код.

Механизм *Look & Feel* позволяет привести вид всех компонентов GUI к привычному виду для ОС, в которой запущено приложение.

**ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №11**

***Цель:***

Научиться разработке сетевых Java приложений с использованием технологии RMI. Разобрать концепцию MUD.

***Предметная область:***

Удаленный вызов процедур. Сетевые приложения..

1. **Краткие теоретические сведения**

Технологии вызова удаленных методов реализованы в пакетах *java.rmi* и *java.rmi.server*. Вызов удаленных методов является мощной технологией для разработки сетевых приложений, освобождающей программиста от необходимости заботиться о деталях реализации сетевых соединений на нижнем уровне.

В этой модели сервер определяет объекты, которые могут использоваться удаленными клиентами. Клиенты вызывают методы удаленных объектов так же, как если бы они были локальными объектами, выполняющимися внутри той же виртуальной машины, что и клиент. Технология RMI скрывает лежащий в ее основе механизм транспортировки параметров методов и возвращаемых значений через сеть. Параметр и возвращаемое значение могут быть либо значением примитивного типа, либо любым сериализуемым объектом.

Для того чтобы создать приложение на базе RMI нужно:

1) Создать интерфейс, расширяющий *java.rmi.Remote*. В этом интерфейсе определены экспортируемые методы, реализуемые удалёнными объектами (то есть методы, реализуемые сервером и вызываемые удаленным клиентом). Каждый метод этого интерфейса должен быть объявлен как генерирующий исключение *java.rmi.RemoteException*, которое является базовым классом других классов исключений RMI. Каждый удаленный метод должен объявлять, что он может сгенерировать *RemoteException* из-за того, что существуют ситуации, приводящие к возникновению ошибок во время процесса вызова удаленных методов через сеть.

2) Определить класс, производный от *java.rmi.server.UnicastRemoteObject*

(или от потомка), реализующий удаленный интерфейс. Этот класс представляет удаленный, или серверный, объект. Кроме объявления того, что его удаленные методы генерируют исключения *RemoteException*, удаленный объект не должен делать что-либо особенное, чтобы позволить вызывать его методы удаленно. Объект *UnicastRemoteObject* (и вся остальная часть инфраструктуры RMI) обрабатывают это автоматически.

3) Написать программу (сервер), создающую экземпляр удаленного объекта. Экспортируйте объект, сделав его доступным для использования клиентами путем регистрации имени объекта в службе реестра. Как правило, это выполняется с помощью класса *java.rmi.Naming* и программы *rmiregistry*. Кроме того, серверная программа может сама выполнять функции сервера реестра, используя класс *LocateRegistry* и интерфейс *Registry* из пакета *java.rmi.registry*.

4) При использовании RMI клиент и сервер не взаимодействуют непосредственно. На стороне клиента ссылка на удаленный объект реализуется в виде экземпляра класса заглушки. Когда клиент вызывает удаленный метод, в действительности вызывается метод объекта-заглушки. Заглушка производит

необходимые сетевые операции по передаче этого вызова находящемуся на сервере классу каркаса. Этот каркас транслирует пришедший по сети запрос в вызов метода серверного объекта, а затем передает возвращенное им значение обратно заглушке, которая, в свою очередь, возвращает его клиенту. Все это представляет собой довольно сложную систему, но прикладным программистам никогда не приходится думать о заглушках и каркасах; они генерируются автоматически утилитой *rmic*. До версии Jаvа 5 заглушки приходилось создавать вручную, а в современных версиях Jаvа эта стадия уже необязательна.

5) Написать клиентскую программу, использующую экспортированный

сервером удаленный объект. Прежде всего, клиенту необходимо получить ссылку на удаленный объект, используя класс *Naming* для поиска объекта по имени. Это имя обычно задается в форме *rmi:URL*. Удаленная ссылка, которая будет возвращена, представляет собой экземпляр интерфейса *Remote* объекта (или, более точно, объект-заглушку удаленного объекта). Как только клиент получил этот удаленный объект, он может вызывать его методы точно так же, как он вызывал бы методы локального объекта. Он должен только знать, что все удаленные методы могут генерировать объекты исключений *RemoteException* и что при возникновении сетевых ошибок это может случаться в самый неожиданный момент.

1. **Задание**
   1. ***Постановка задачи:***

* Изучить пример 2
* Проанализировать вариант задания. Можно ли его реализовать как часть MUD системы (например, в одной из комнат MudPlace), требуется ли для этого внести изменения в парадигму MUD? Какие изменения потребует реализация клиента MUD, другие классы примера? Оформить эти размышления в отчёте в качестве анализа предметной области. При реализации, по возможности, использовать парадигму MUD и классы примера 2 при реализации варианта задания.
* Создать на основе технологии RMI клиент/серверное приложение.

Чат. Клиент регистрируется на сервере при подключении, сервер сообщает всем о подключении/отключении клиентов. Клиент посылает через сервер сообщение, которое получают все клиенты, подключенные в данный момент

* 1. ***Решение задачи:***

Для решения задачи были использованы классы из примера. Сервер реализован как MUD с одной комнатой, где находятся все клиенты чата. Клиенты чата могут отправлять сообщения только лично друг другу. В связи с этим классы из примера были упрощены.

Клиент реализован как графическое приложения. У каждого клиента есть поле для ввода сообщений, список подключенных пользователей (кроме себя) и поле с выводом служебной информации и сообщений от других пользователей. Подключение реализовано как простой модальный диалог для ввода информации.

Для запуска необходимо заранее запустить *rmiregistry*. После запускается сервер с указанием названия сервера. После этого клиенты могут подключаться к серверу.

**2*.3. Результат:***

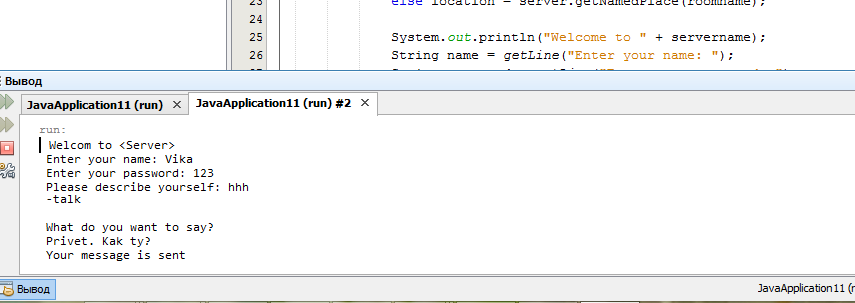


Рисунок 11.1 – Результат работы приложения

1. **Выводы**

RMI позволяет разрабатывать сетевые Java приложения, не используя низкоуровневые сетевые интерфейсы.

Для регистрации классов RMI на сервере должен быть запущен *rmiregistry*.

В случае проблем с сетью, удаленным вызовом и т.п. методы выбрасывают исключение *RemoteException*.

Классы для использования в RMI должны наследоваться от *UnicastRemoteObject*.

**ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №12**

***Цель:***

Научиться использованию нативного C/C++ кода из написанных на Java приложений.

***Предметная область:***

Разработка библиотек.

1. **Краткие теоретические сведения**

В языке Java при помощи технологии JNI (Java Native Interface) можно использовать функции, написанные на языках более низкого уровня – C/C++ и asm. Методы, которые представляют эти функции, указываются с ключевым словом *native*.

Для генерации заголовков функции для библиотеки можно использовать утилиту *javah*, либо создавать их вручную. Для таких библиотек подключается заголовочный файл *jni.h*, в котором определены все типы и функции для работы со средой.

Каждая функция библиотеки, кроме параметров, указанных разработчиком, добавляются еще два – интерфейс для обращения к среде Java и объект класса, в котором содержится нативный метод. Интерфейс позволяет вызывать методы языка Java из языка низкого уровня, создавать объекты, совершать конвертации типов, получать доступ в полям класса.

После реализации всех функций, библиотека компилируется как обычная библиотека Win32. Перед тем, как использовать ее в среде Java, ее необходимо подгрузить при помощи метода *System.loadLibrary*, при этом библиотека должна иметь ту же разрядность, что и среда *JVM*.

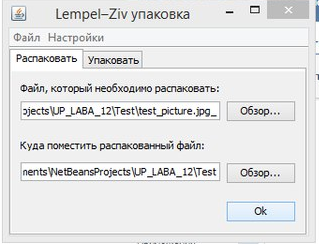
1. **Задание**
   1. ***Постановка задачи:***

Реализовать работу с упакованными файлами Windows (см. функции Win32-API: LZOpenFile, LZCopy, LZClose).

* 1. ***Решение задачи:***

При решении используется класс PackClass, в котором объявляются два статических нативных метода: на сжатие и распаковку. Методы принимают в себя путь к файлу, а также путь, куда нужно поместить сжатый/извлеченный файл. При сжатии используется утилита Windows compress.exe. Путь к ней указывается в меню приложения. Библиотека загружается в классе в статическом блоке. Используется при этом функция System.LoadLibrary (string). Функция грузит dll, а затем можно свободно вызывать подгруженные методы. В библиотеке функции объявляются в .h файле, определяются – в .cpp файле.

**2*.3. Результат:***

****

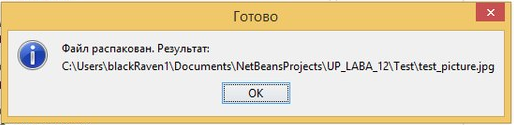
****

Рисунок 12.1 – Результат работы приложения

1. **Выводы**

Нативные методы достаточно удобно использовать: нужно всего лишь указать путь к библиотеке, а затем загрузить ее.

Нативные методы очень просто вызываются – как обычные методы класса.

**ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №13**

***Цель:***

Научиться разработке сервлетов. Разобраться в принципах работы с Apache Tomcat.

***Предметная область:***

Сетевые приложения. WEB-разработка.

1. **Краткие теоретические сведения**

Технология Java не могла пройти мимо такой насущной потребности и отозвалась на нее созданием сервлетов и языком JSP (JavaServer Pages).

Сервлеты (servlets) выполняются под управлением Web-сервера подобно тому, как апплеты выполняются под управлением браузера, откуда и произошло их название.

Для слежения за работой сервлетов и управления ими создается специальный программный модуль, называемый контейнером сервлетов (servlet container).

Контейнер сервлетов загружает сервлеты, инициализирует их, передает им запросы клиентов, принимает ответы. Сервлеты не могут работать без контейнера, как апплеты не могут работать без браузера.

Web-сервер, снабженный контейнером сервлетов и другими контейнерами, стал называться сервером приложений (application server, AS).

Чтобы сервлет мог работать, он должен быть зарегистрирован в контейнере, установлен (deploy) в него. Установка (deployment) сервлета в контейнер включает получение уникального имени и определение начальных параметров сервлета, запись их в конфигурационные файлы, создание каталогов для хранения всех файлов сервлета и другие операции.

Процесс установки сильно зависит от контейнера. Одному контейнеру достаточно скопировать сервлет в определенный каталог, например autodeploy/ или webapps/, другому надо после этого перезапустить контейнер, для третьего надо воспользоваться утилитой установки. В стандартном контейнере Java EE SDK такая утилита называется deploytool.

Один контейнер может управлять работой нескольких установленных в него сервлетов. При этом один контейнер способен в одно и то же время работать в нескольких виртуальных машинах Java, образуя распределенное Web-приложение. Сами же виртуальные машины Java могут работать на одном компьютере или на разных компьютерах.

Контейнеры сервлетов создаются как часть Web-сервера или как встраиваемый в него модуль. Большую популярность получили встраиваемые контейнеры Tomcat, разработанные сообществом Apache Software Foundition

1. **Задание**
   1. ***Постановка задачи:***

* Изучите наиболее общую информацию в файле Java-УП-14.ppt.
* Прочитайте статью по http и сервлетам по адресу http://www.javaportal.ru/java/articles/java\_http\_web/article05.html.
* Прочитайте статью по JSP по адресу http://www.codenet.ru/webmast/java/jsp.php
* Установите на ваш персональный компьютер Apache Tomcat 8.\* и Ant 1.\* (При инсталляции Tomcat, выберите установку примеров – Examples!).
* Перейдите на страницу примеров сервлетов TomCat <http://127.0.0.1:8080/examples/servlets/>. Изучите работу предлагаемых примеров и их исходный код.
* Перейдите на страницу примеров JSP TomCat <http://127.0.0.1:8080/examples/jsp/>. Изучите работу предлагаемых примеров и их исходный код.
* Изучите части 1-2 книги “Java Servlet & JSP Cookbook”.
  1. ***Решение задачи:***

Перед использованием были изучены все приведенные материалы. После этого были установлены Apache Tomcat 8.\* и Ant 1.\* После этого были рассмотрены некотрые примеры простых сервлетов, взаимодействующих с HTML.

**2*.3. Результат:***

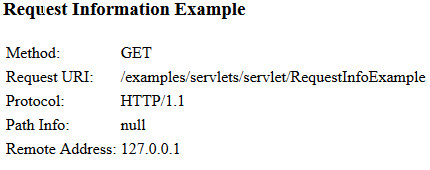


Рис. 13.1 – Скриншот примера с сервлетами

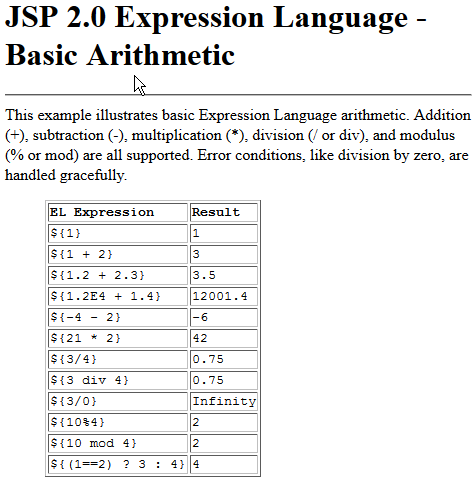


Рис. 13.2 – Скриншот примера с JSP

1. **Выводы**

В качестве веб-сервера для сервлетов можно использовать бесплатный Apache Tomcat.

При помощи Java можно создавать сервлеты, при помощи которых можно создавать динамические веб-сайты.

При разработке веб-приложений можно использовать как сервлеты, так и технологию JSP (JavaServer Pages).