**Министерство образования Республики Беларусь**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики и информатики**

Кафедра информационных систем управления

**ОТЧЕТ**

По дисциплине:

«Учебная практика»

Выполнила

студентка гр. 9

Шелег Ф. В.

Проверил

Высоких Л. К.

Минск 2015

СОДЕРЖАНИЕ

1 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №1………………….3

2 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №2………………….5

3 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №3………………….7

4 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №4………………….9

5 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №5………………...12

6 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №6………………...14

7 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №7………………...16

8 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №8………………...18

9 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №9………………...20

10 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №10………………22

11 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №12………………24

12 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №13………………25

13 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №14………………28

ПРИЛОЖЕНИЕ A

A.1 Исходные коды лабораторной работы №1……………………………….30

A.2 Исходные коды лабораторной работы №2……………………………….56

A.3 Исходные коды лабораторной работы №3……………………………….64

A.4 Исходные коды лабораторной работы №4……………………………….74

A.5 Исходные коды лабораторной работы №5……………………………….94

A.6 Исходные коды лабораторной работы №6……………………………...103

A.7 Исходные коды лабораторной работы №7……………………………...116

A.8 Исходные коды лабораторной работы №8……………………………...120

A.9 Исходные коды лабораторной работы №9……………………………...130

A.10 Исходные коды лабораторной работы №10……………………………..134

…………

A.11 Исходные коды лабораторной работы №12……………………………..155

A.12 Исходные коды лабораторной работы №13……………………………..156

A.13 Исходные коды лабораторной работы №14……………………………..159

**1** **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.

**2** **ЗАДАНИЕ**

**2.1 Постановка задачи:**

Создать класс, реализующий интерфейс Shape, для изображения стрелки в плоскости экрана;

С помощью аффинного преобразования координат изобразить вращение стрелки вокруг произвольной точки. Фигура перемещается с постоянной скоростью;

Рисунок выполнить в окне апплета или фрейма с выбранной толщиной границы фигуры, цветами границы и фона и точки, вокруг которой вращается стрелка (толщина, цвет и координаты точки вводятся в качестве аргументов программы или параметров апплета).

**2.2 Решение задачи:**

Для отрисовки овала создается класс-наследник класса Line2D.Double, который в свою очередь реализует интерфейс Shape. Добавляется метод draw, отвечающий за непосредственную отрисовку овала. Также используются дополнительные поля, отвечающие за изменение угла наклона при аффинных преобразованиях. В апплете используется специальный метод, преобразовывающий 3 числа в цвет (формат RGB). Для плавности работы рисунок выполняется на панели.

**2.3 Результат:**

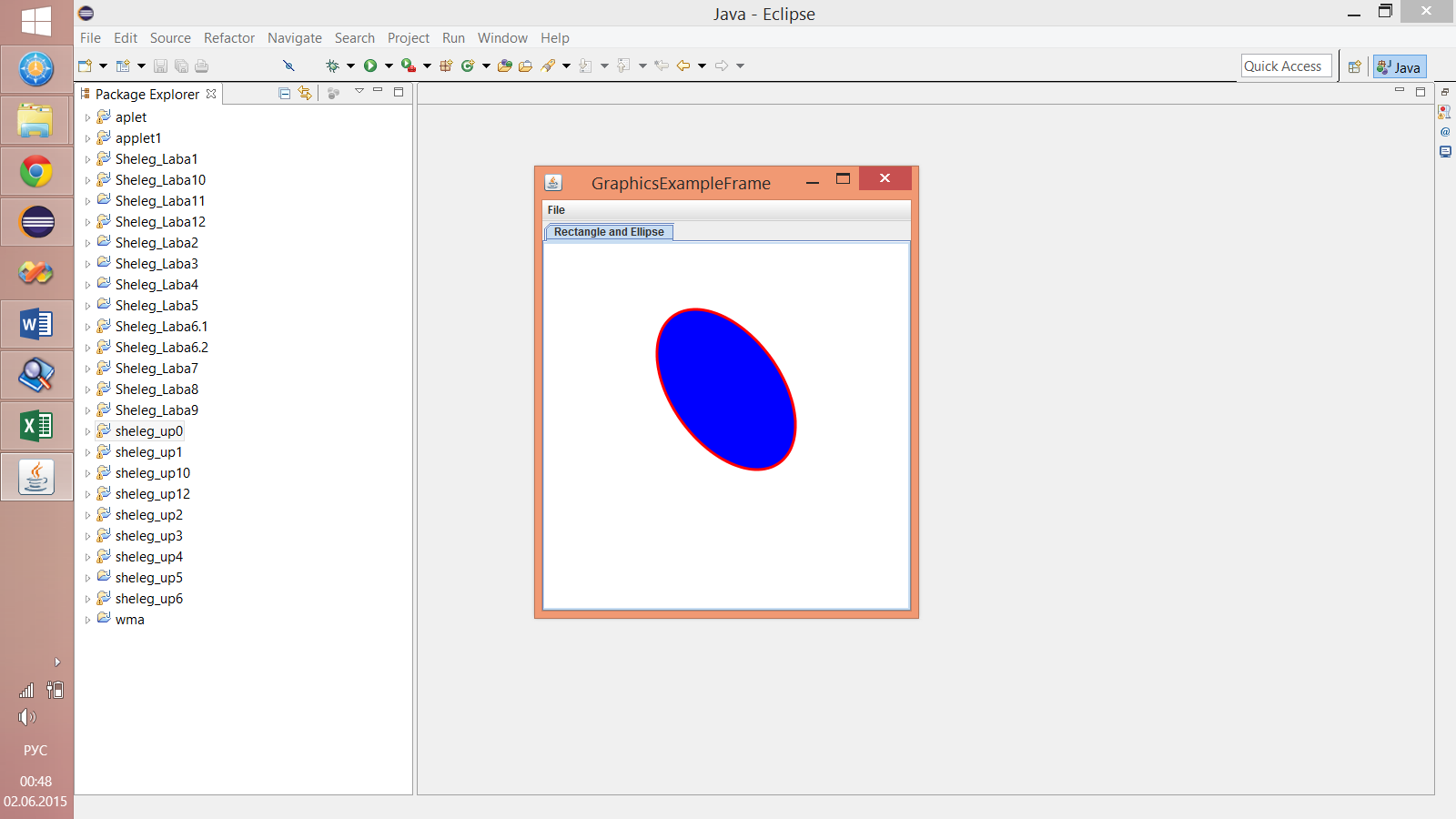


Рисунок 1.1 – Скриншот окна приложения

**3** **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.

**2** **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,
* 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**

 Фигура (дорожный знак): надпись STOP в прямоугольнике, цвет прямоугольника и   
надписи – красный, цвет фона – серый с градиентной заливкой снизу-вверх  
Фильтр: Rotate CW 45 degrees (поворот по часовой стрелке на 45)

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.

**2.2 Решение задачи**

Используется класс-наследник класса Rectangle2D.Double. Для отрисовки объектов этого класса используется метод draw, принимающий параметры Graphics2D и координаты x и y типа double для корректировки расположения фигуры. Размер символа корректируется в этом методе под размеры квадрата.

Тень от знака рисуется при помощи аффинных преобразований.

Градиентная заливка устанавливается в методе setPaint, принимающем объект класса GradientPaint. Фильтр представлен объектом класса BufferedImageOp.

В программе используется один внеэкранный буфер, в созданную графику которого рисуется фильтрованный знак. Затем этот буфер копируется в стандартную графику апплета методом drawImage, который, собственно, и получает этот буфер в качестве параметра фильтра.

Для установки сглаживания к графике применяется метод setRenderingHint.

**2.3 Результат:**

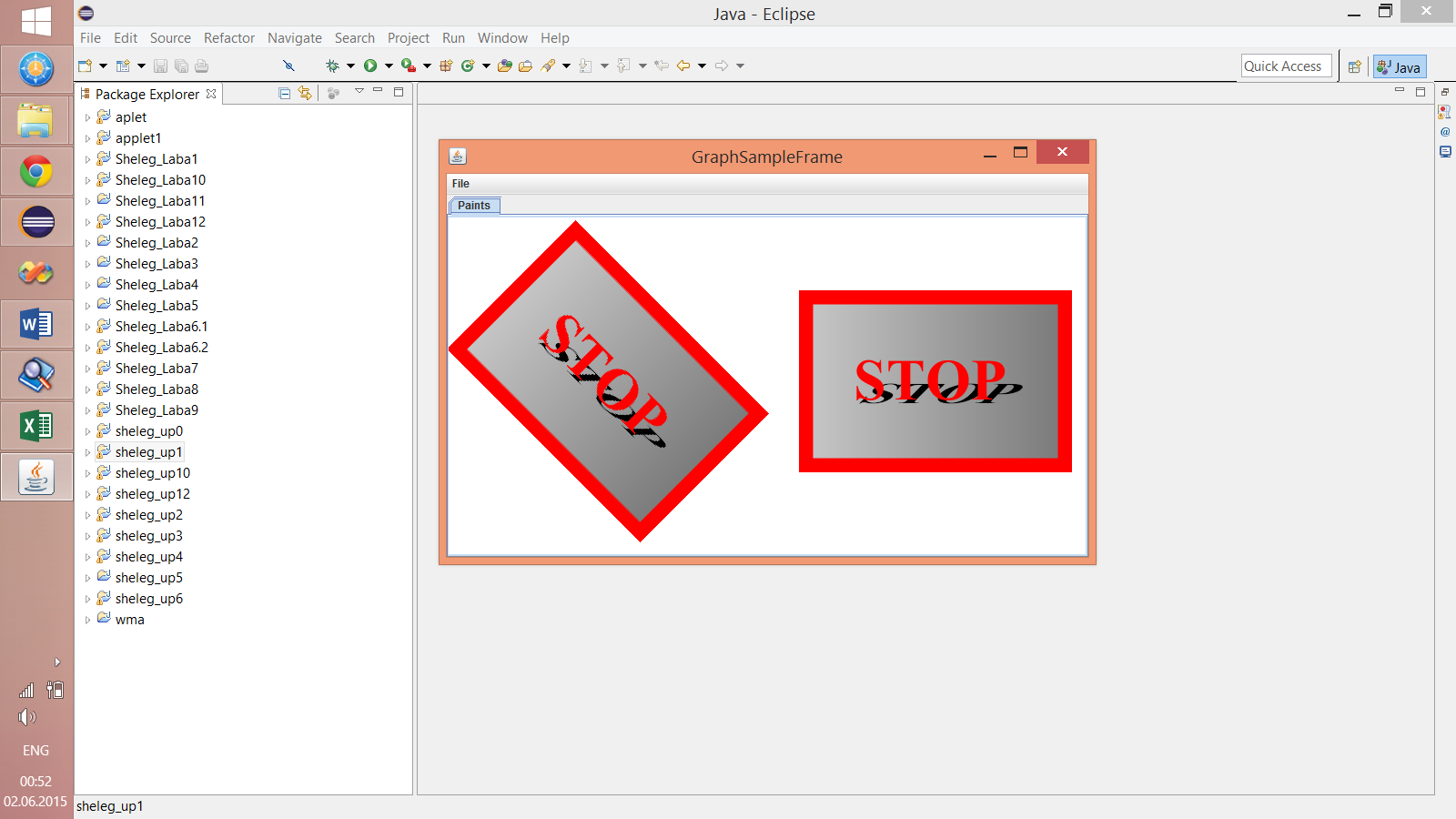


Рисунок 2.1 – Результат работы приложения

* 1. **ВЫВОДЫ**
* Работа с внеэкранными буферами незначительно отличается от работы напрямую. Необходимо только воспользоваться методом BufferedImage.getGraphics(), после чего работать с привычным классом Graphics. Затем этот буфер можно легко перенести в любую графику.
* Множество фильтров обработки изображений уже реализованы в Java, и они достаточно просты в использовании.
* Градиентной заливкой легко управлять изменением координат.
* Для рисования теней удобно пользоваться аффинными преобразованиями, которые позволяют нарисовать тень под любым углом. Стоит только лишь реалистично рассчитать угол падения тени.

**3** **ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №3**

**Цель:**

Научиться создавать и использовать пользовательские классы Stroke, пользовательские фигуры, переопределять их методы.

**Предметная область:**

Двумерная графика.

**1 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Класс Stroke преобразует операцию рисования линий в операцию заливки области, принимая объект Shape, очертания которого нужно прорисовать, и возвращая штриховую фигуру, представляющую собой контур исходной фигуры. Поскольку интерфейс Stroke прост, то относительно нетрудно определять собственные классы, реализующие интерфейс Stroke и создающие интересные графические эффекты.

Интерфейс Shape. Интерфейс Shape определяет три важных метода:

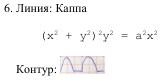
Contains() – определяет, содержит ли наша фигура точку или прямоугольник;

Intersects() – определяет, пересекается ли какая-то часть фигуры с заданным прямоугольником;

getPathIterator() – возвращает объект PathIterator, описывающий контур фигуры в терминах сегментов линий и кривых. Java 2D использует объекты PathIterator при рисовании и заполнении фигур.

* 1. **ЗАДАНИЕ**

1. **Постановка задачи**

Нарисовать линию циссоиды, используя пользовательский класс Shape, реализующий рисование указанной линии. Разработать пользовательский класс Stroke для отрисовки контура линии, указанного в задании. Для демонстрации и тестирования создать приложение или апплет.

**2.2 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

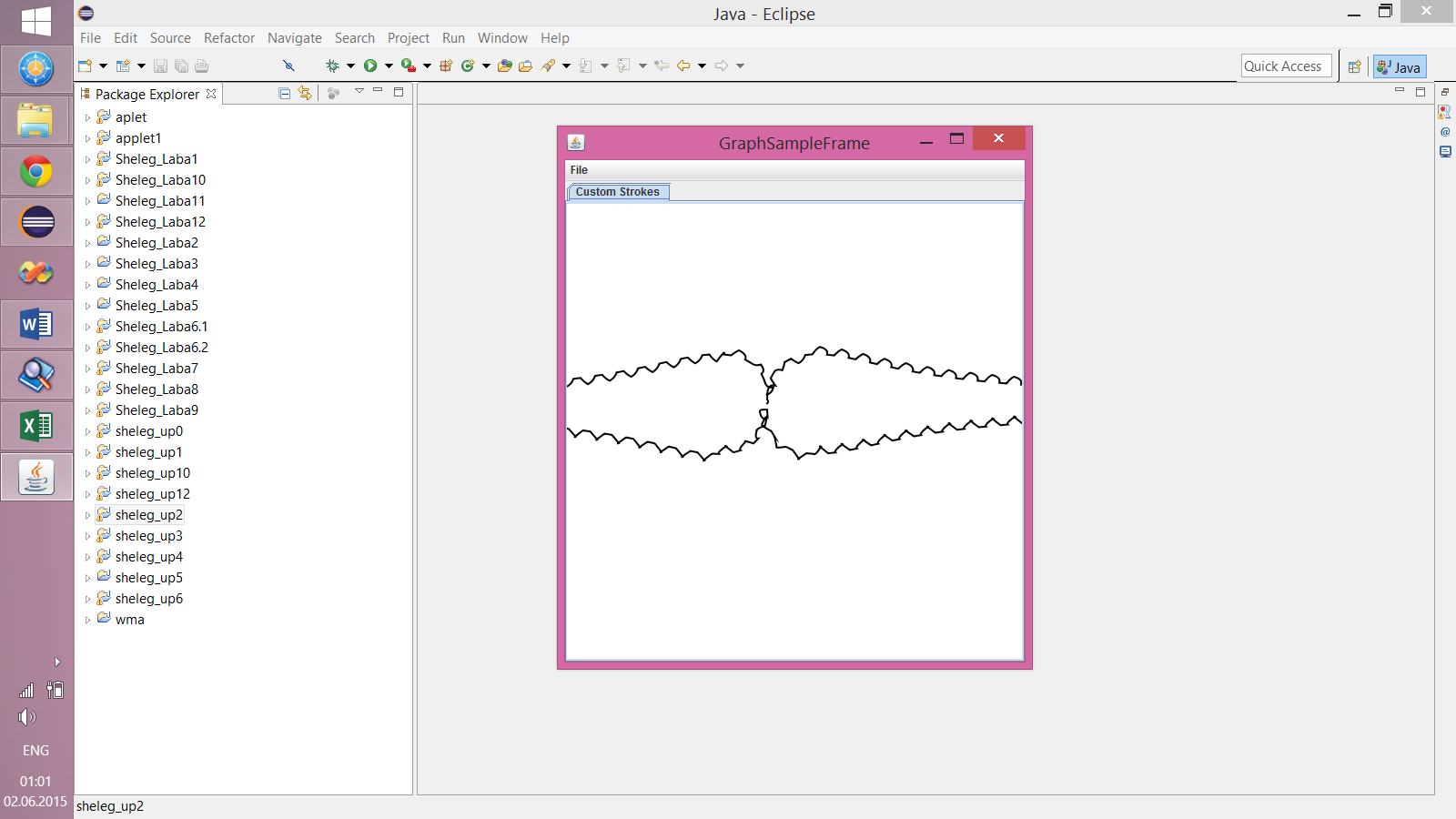
**2.3 Результат:**

Рисунок 3.1 – Результат работы приложения

**3 ВЫВОДЫ**

Для создания фигуры любого задуманного вида можно использовать пользовательский класс, реализующий Shape;

Контур отрисовки аналогично представляется классом, реализующим Stroke;

При переопределении метода getPathIterator() класса пользовательской фигуры удобно и необходимо создать класс, реализующий интерфейс PathIterator;

Для незамкнутых фигур не нужно переопределять метод contains().

**4** **ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №4**

**Цель:**

Научиться работать с принтером в Java, а также подготавливать текст и изображения для последующей отправки на печать, устанавливать параметры печати, отправлять на печать документы.

**Предметная область:**

Печать документов.

**1 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

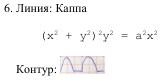
Для печати документов в Java используются несколько основных классов. Это класс PrinterJob, который управляет печатью. С его помощью можно вызвать диалоговое окно печати с пользователем и настроить соответствующие свойства.

Классы, реализующие интерфейс Printable, используются для представления страниц распечатываемых документов.

Классы, реализующие интерфейс Pageable, используются для представления многостраничных документов. Он определяет три метода, которые возвращают количество страниц в документе, размер и ориентацию каждой страницы документа, объект Printable, отвечающий за соответствующую страницу.

* 1. **ЗАДАНИЕ**

1. **Постановка задачи**



Модифицировать предыдущее задание (нарисовать линию с контуром, указанные выше), добавив возможность печати небольшого отчета о решении задания №3. Отчет должен содержать рисунок из предыдущего задания и исходный текст класса Shape, реализующий рисование указанной алгебраической линии.

При печати использовать режим альбомной ориентации страницы и двустороннюю печать. При этом рисунок должен занимать не более половины страницы и быть выровненным по горизонтали.

**2.2 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

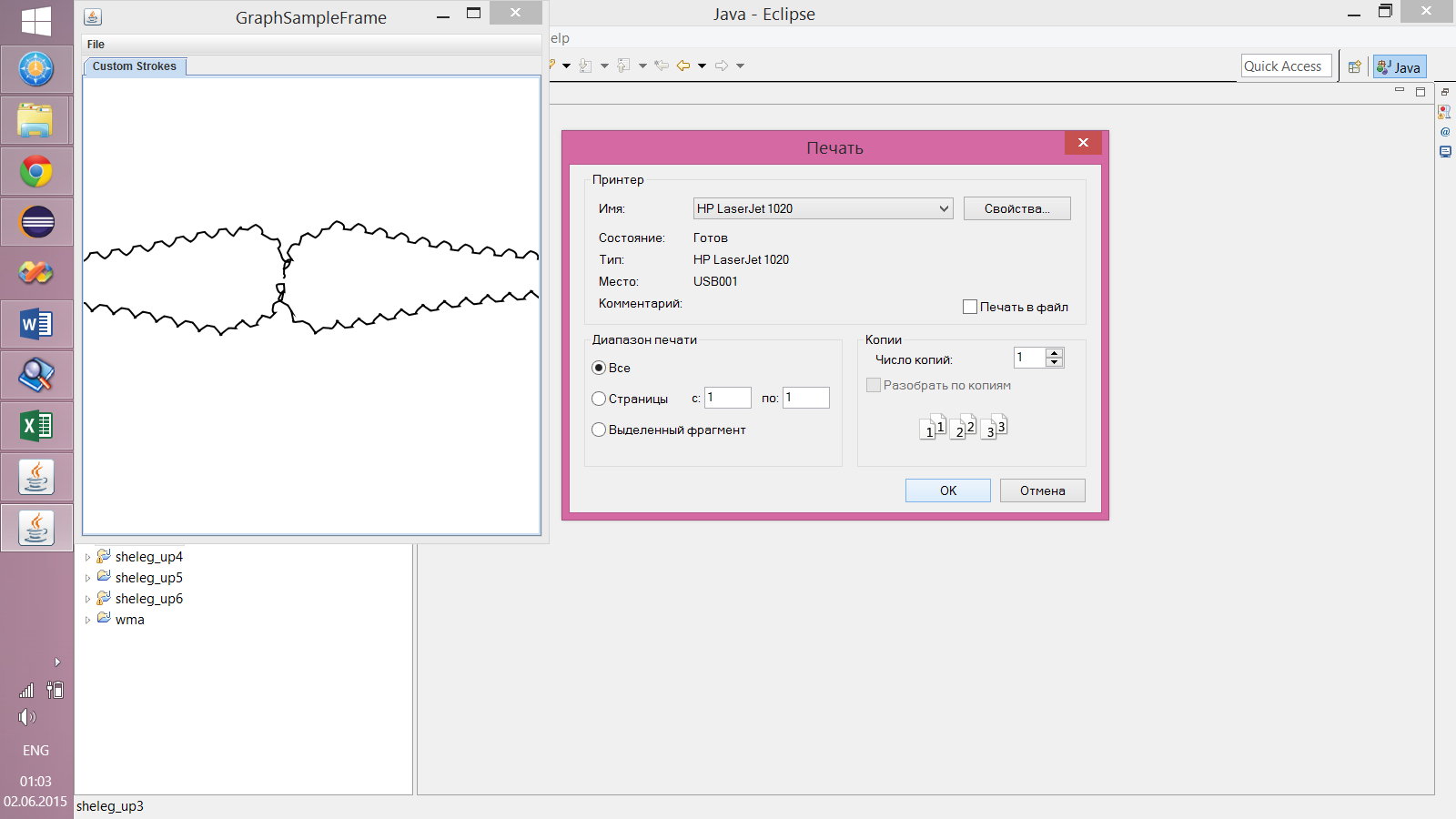
результат, который будет соответствовать координате y. Также имеются два поля, ограничивающие область задания функции. Метод getPathIterator() возращает объект внутреннего класса LineIterator, который содержит методы, необходимые для задания сегментов для отрисовки.

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.

**2.3 Результат**

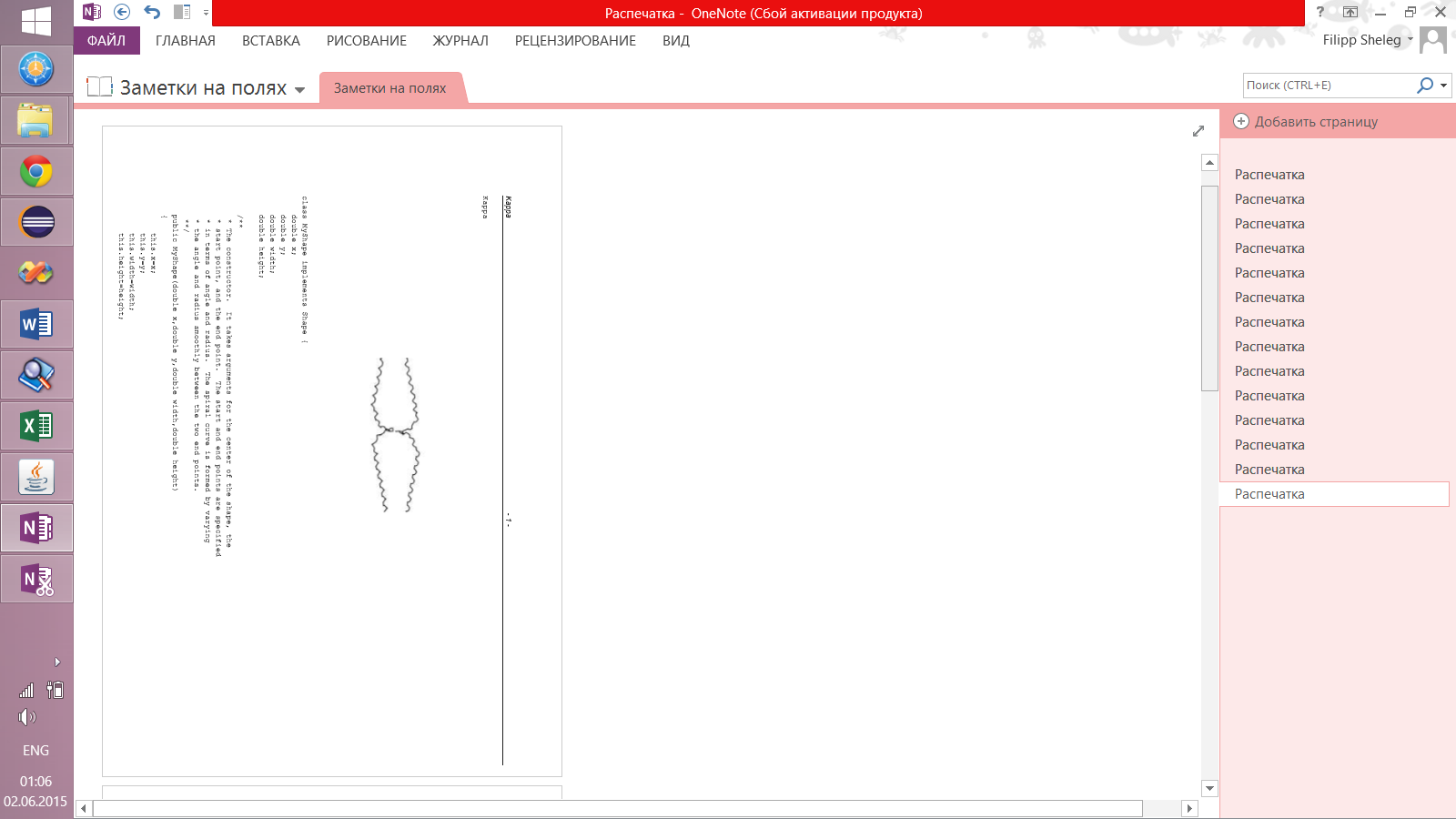


Рисунок 4.1/2 – Процесс/Результат работы приложения

**3 ВЫВОДЫ**

Печатать текст и выводить графику на принтер можно, используя объект класса Graphics, как и с обычным экраном.

Разрывы страниц и все необходимые расчеты мы должны выполнять сами.

С одной стороны, это может занять лишнее время, с другой – позволяет сделать отчет необходимого вида и размера, как следствие, добавляя ему гибкости.

**5** **ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №5**

**Цель:**

Научиться обмену объектами в компонентах Swing средствами пакетов *java.awt.datatransfer* и *java.awt.dnd* в видеdrag-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,

* тип MIME, определяющий кодировку, используемую при передаче данных.
  + 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 онможет использовать для передачи своих данных;

*isDataFlavorSupported()* –проверяет,поддерживает ли объект *Transferable* данный

формат

*getTransferData()* –возвращает данные в формате,соответствующемзапрошенному *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.

**2** **ЗАДАНИЕ**

**2.1 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).

**2.2 Решение задачи:**

Класс *Graph*, который представляет нашу фигуру, теперь реализует интерфейсы

*Transferable* и *Serializable*.

Класс *DrawComponent* содержит в себе коллекцию графиков, отвечает за механизм drag-and-drop для графиков, их перемещение.

Класс *MainFrame* представляет собой главный фрейм с объектом класса

*DrawComponent* и меню(File, View, Help).

Drag-and-drop работает как в пределах одного приложения, так и в пределах нескольких. При зажатой клавише Ctrl вместо перемещения кривой происходит ее копирование.

Фигуры можно выделять нажатием по ним, после чего они меняют свой цвет. Эти цвета можно изменить в меню View. Выделенную фигуру можно передвигать, используя стрелки.

При нажатии на форму с зажатой клавишей Alt добавляется новая кривая на компонент.

Форму можно очистить как через меню File, так и используя сочетание клавиш

Ctrl+N.

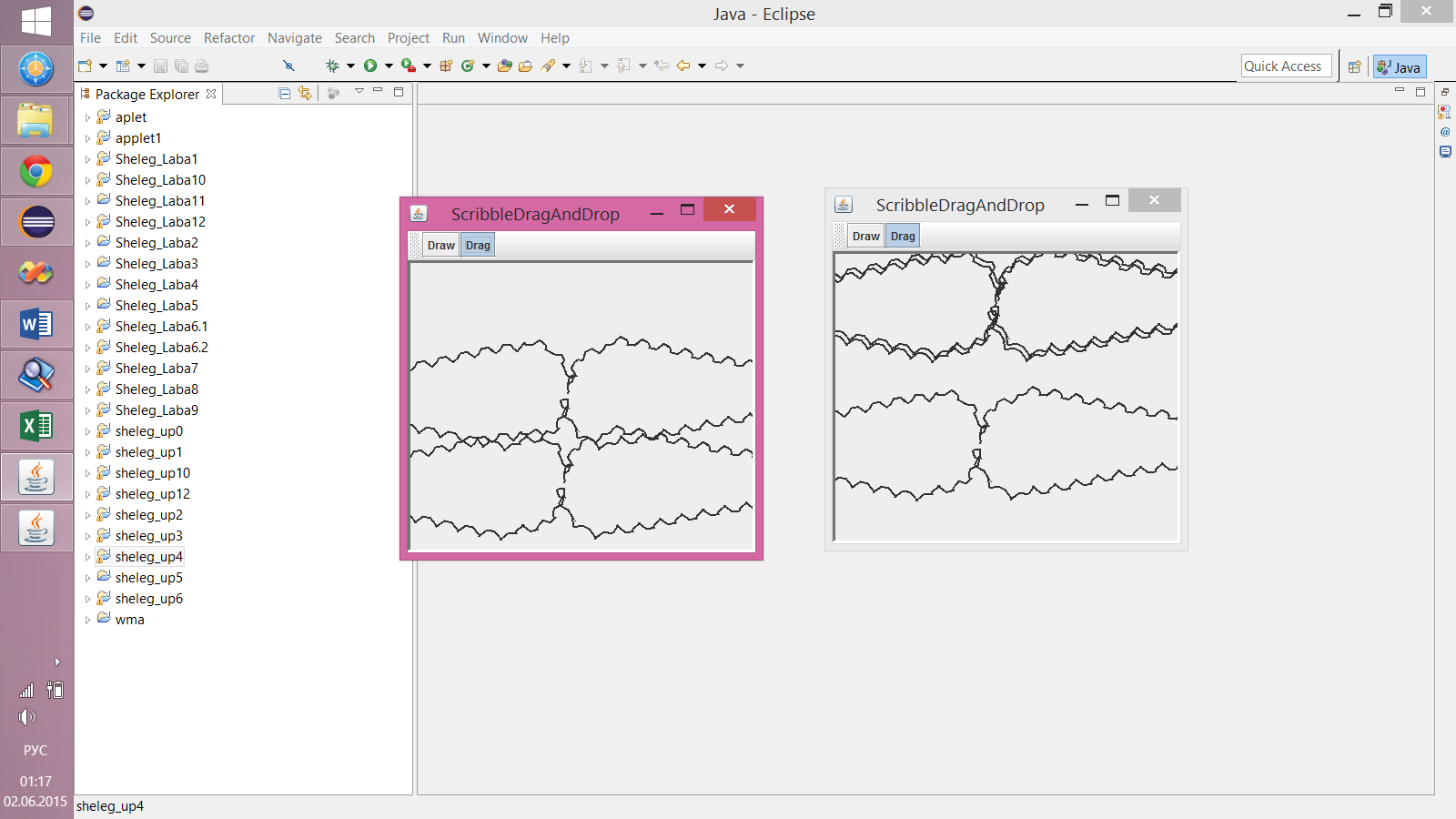
**2.3 Результат**

Рисунок 5.1 – Вид итогового приложения

**3** **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

**6** **ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №6**

**Цель:**

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

**Предметная область:**

Каталог исторических памятников.

**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*.

**2 ЗАДАНИЕ**

**2.1 Постановка задачи:**

Разработать систему классов/интерфейсов для расписания автовокзалов. 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).

**2.2 Решение задачи:**

На главной форме находятся дерево, при помощи которого можно осуществлять фильтрацию элементов, и таблица, в которой отображаются записи. Таблица поддерживает сортировку. При выборе корневого элемента в дереве, отображаются все элементы. При выборе фильтра в дереве, отображаются только элементы, соответствующие фильтру. Фильтры добавляются динамически при добавлении маршрутов.

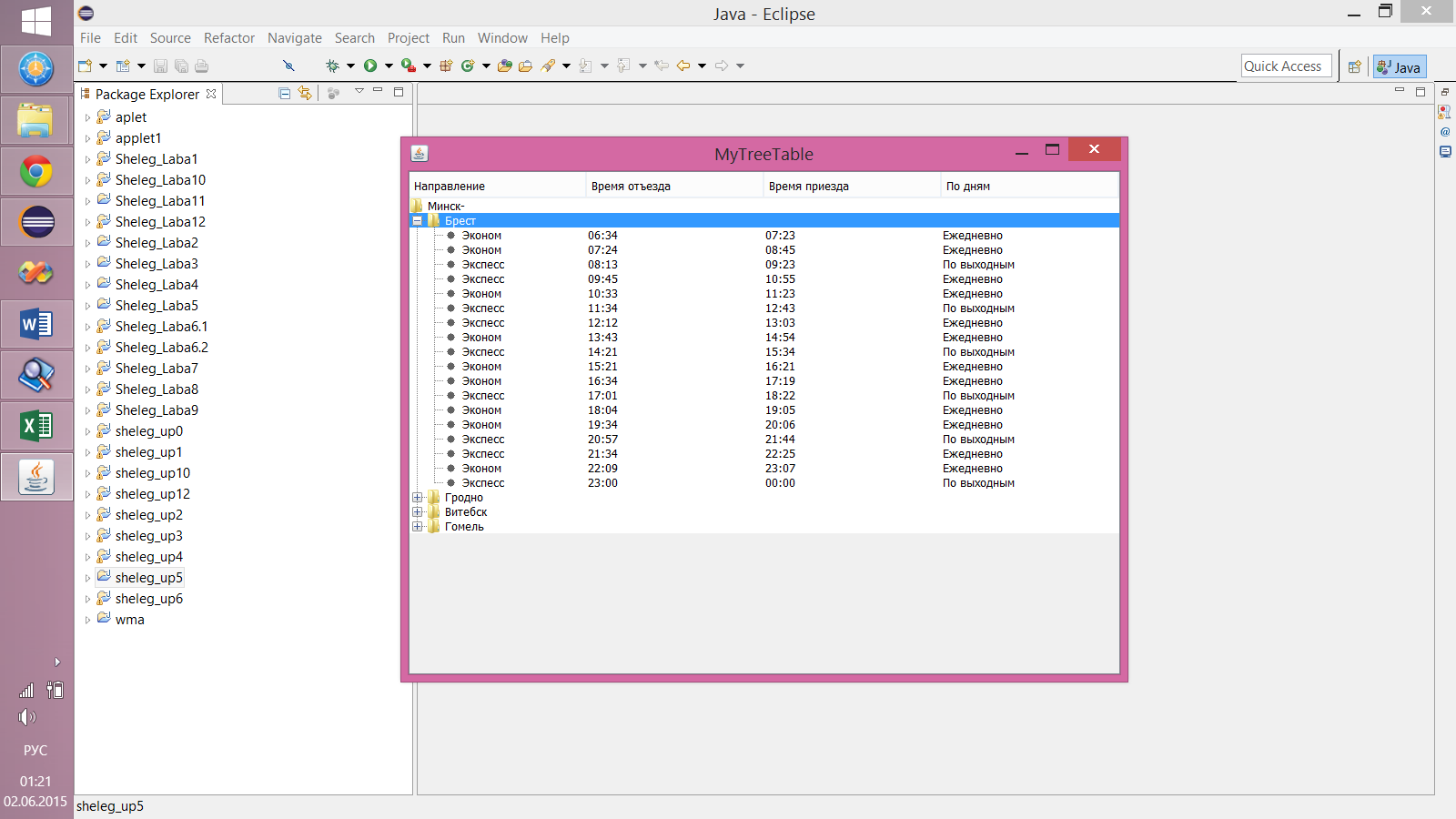
**2.3 Результат:**

Рисунок 6.1 – Результат работы программы

**3 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 *TableModel* и *TreeModel*.

**7** **ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №7**

**Цель:**

Научиться создавать собственные java-bean компоненты, а также устанавливать их.

**Предметная область:**

Двумерная графика, JavaBeans.

* 1. **КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**
* Программный интерфейс JavaBeans предоставляет среду для разработки многократно используемых, встраиваемых, модульных программных компонентов.
* Любой объект, удовлетворяющий определенным базовым правилам и соглашениям об именах, может считаться компонентом JavaBeans. Не существует никакого класса Bean, который все компоненты должны были бы иметь в качестве базового.
* Для того чтобы подготовить компонент к использованию в контейнере, вы должны упаковать его вместе с необходимыми ему файлами и ресурсами в файл JAR.

**2 ЗАДАНИЕ**

1. **Постановка задачи**

Изучить материал примера по быстрому введению в среду разработки NetBeans и компоненты JavaBeans по адресу http://docs.oracle.com/javase/tutorial/javabeans/quick/index.html.

Разработать компонент «эллиптическая заливка» на базе класса Canvas. Свойства: ширина, высота и цвет. Создать файл манифеста и упаковать компонент вместе с исходным кодом разработанных классов.

**2.2 Решение задачи**

Для представления вертикальной линии был создан класс Ellipse. В этом классе имеются свойства: высота, цвет. Также имеются два конструктора и геттеры и сеттеры для свойств. Для отрисовки линии был переопределен метод paint(Graphics g). В файле манифеста содержится следующая информация:

Name: Ellipse.class

Java-Bean: True

Созданный компонент упаковывается в .jar-архив (up-7.jar). Чтобы использовать его, нужно установить его в JavaBeans.

**2.3 Результат:**

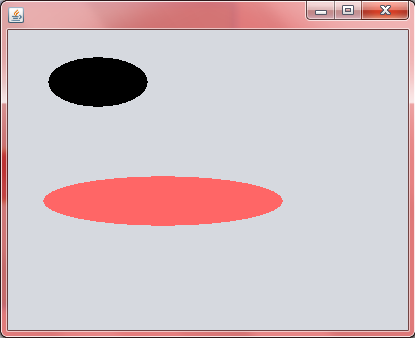


Рисунок 7.1 – Запуск фрейма

* 1. **ВЫВОДЫ**
* Все Swing- и Awt- компоненты могут функционировать как компоненты JavaBeans.
* Процедура установки компонента зависит от используемого контейнерного средства.

- Модель JavaBeans обеспечивает взаимодействие с аналогичными компонентными средами. Так, например, обычная Windows программа может с помощью соответствующего моста или компонента-обертки пользоваться компонентом Java так, как будто он является компонентом

COM или ActiveX.

**8** **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.

**2 ЗАДАНИЕ**

**2.1 Постановка задачи:**

Разработать компонент: однострочный статический текст, строка ввода и кнопка; свойствами являются текст и текст кнопки. Создать файл манифеста и упаковать компонент. При разработке поместить все классы в пакет *bsu.fpmi.educational\_practice2015*. 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.

**2.2 Решение задачи:**

В качестве компонента *JavaBeans* выступает класс *YesNoPanel*. 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 *YesNoPanelBeanInfo*. Большая его часть была автоматически сгенерирована средой

*NetBeans*.

Был создан jar архив для интеграции компонента в *NetBeans*.

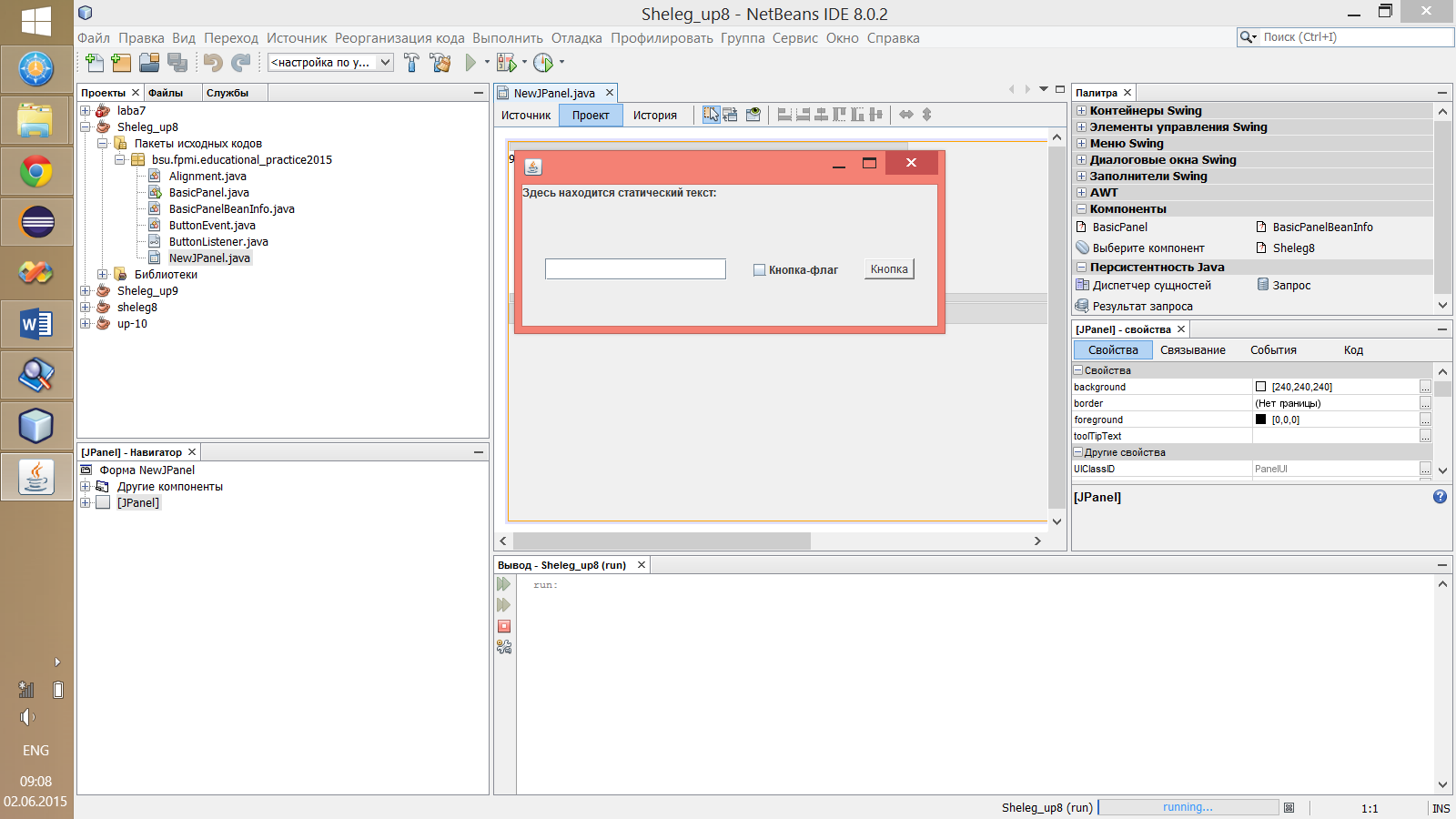
**2.3 Результат:**

Рисунок 8.1 – Окно демонстрационного приложения

**3 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*.

**9** **ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №9**

**Цель:**

Научиться разрабатывать собственные редакторы свойств и настройщики компонентов JavaBeans.

**Предметная область:**

JavaBeans.

**1 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Компонент может предоставлять используемые контейнерным средством 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 использовании простого соглашения об именах. Если ваш тип определен в классе X, редактор для него должен быть определен в классе XEditor.

Другим возможным способом является регистрация редактора свойств путем вызова метода PropertyEditorManager.registerEditor(), например, из конструктора вашего 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.

**2 ЗАДАНИЕ**

**2.1 Постановка задачи**

Создать собственный редактор для каждого свойства компонента из лабораторной работы №8. Каждый редактор должен ограничивать возможные значения свойства, предоставляя выбор из списка трёх – пяти допустимых значений. Зарегистрировать редакторы в классе BeanInfo компонента.

**2.2 Решение задачи**

* данном проекте есть класс AlignmentEditor, который есть наследник класса PropertyEditorSupport. Данный класс предназначен для выбора свойств надписей на панели. На выбор предоставляются следующие варианты свойств: положение текста слева, положение текста справа, а также – по центру. Не менее важным классом является YesNoPanelCustomizer – наследник класса Panel, который позволяет редактировать свойства компонента в режиме реального времени.

**2.3 Результат:**

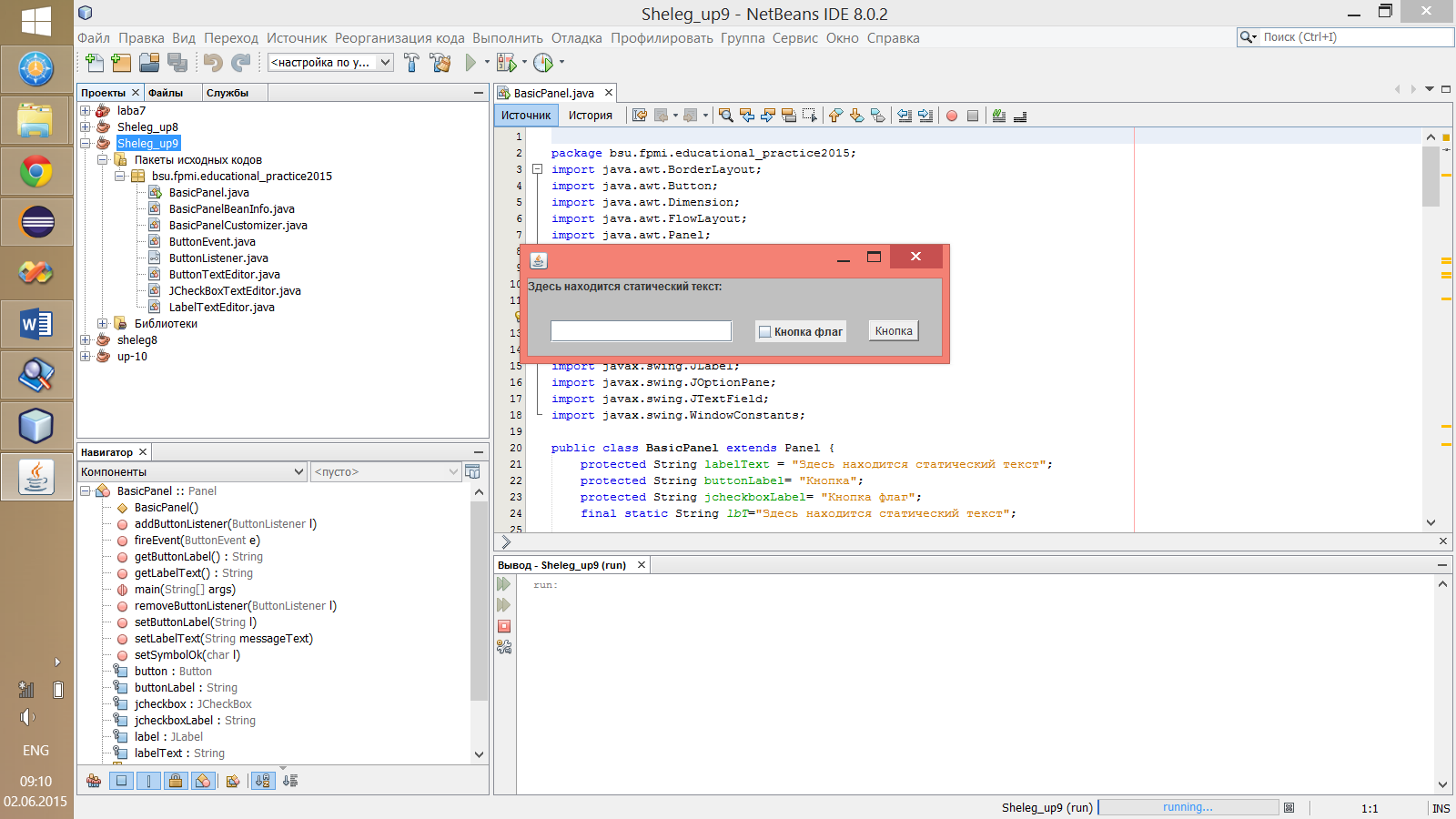


Рисунок 9.1

* 1. **ВЫВОДЫ**
* Вместо непосредственной реализации интерфейса PropertyEditor можно создавать классы, производные от PropertyEditorSupport.
* Свойству, которое имеет перечисляемые значения, необходим простой редактор

свойств.

- Методы PropertyEditor позволяют определить три способа отображения значения свойства и два способа редактирования значения свойства пользователем.

**10 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №10**

**Цель:**

Научиться работать со свойствами фрейма, а также обрабатывать события. Работа с текстом.

**Предметная область:**

Фрейм и его свойства, работа с текстом.

**1 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Класс Stroke преобразует операцию рисования линий в операцию заливки области, принимая объект Shape, очертания которого нужно прорисовать, и возвращая штриховую фигуру, представляющую собой контур исходной фигуры. Поскольку интерфейс Stroke прост, то относительно нетрудно определять собственные классы, реализующие интерфейс Stroke и создающие интересные графические эффекты.

Интерфейс Shape. Интерфейс Shape определяет три важных метода:

Contains() – определяет, содержит ли наша фигура точку или прямоугольник;

Intersects() – определяет, пересекается ли какая-то часть фигуры с заданным прямоугольником;

getPathIterator() – возвращает объект PathIterator, описывающий контур фигуры в терминах сегментов линий и кривых. Java 2D использует объекты PathIterator при рисовании и заполнении фигур.

**2 ЗАДАНИЕ**

1. **Постановка задачи**

Разработать приложение для выполнения несложных действий с текстовым файлом.

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

При реализации интерфейса следовать рекомендациям стандарта CUI. Главное меню должно содержать команды завершения приложения после подтверждения пользователя, выбор шрифта для компонентов интерфейса, выбор стиля интерфейса, вывод информации о разработчике.

**2.2 Решение задачи**

Главное меню приложения содержит 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

Help/About– вывод диалогового окна с информацией

При открытии файла срабатывает функция, которая отображает текст файла в панели слева, и если текст содержит слова с заглавной буквой, то они отображаются в виде кнопок в правой панели окна, а при нажатии на них меняется заголовок окна на соответствующий кнопке текст.

Все действия в приложении реализуются как обработка событий, поступивших в результате нажатий на те или иные кнопки.

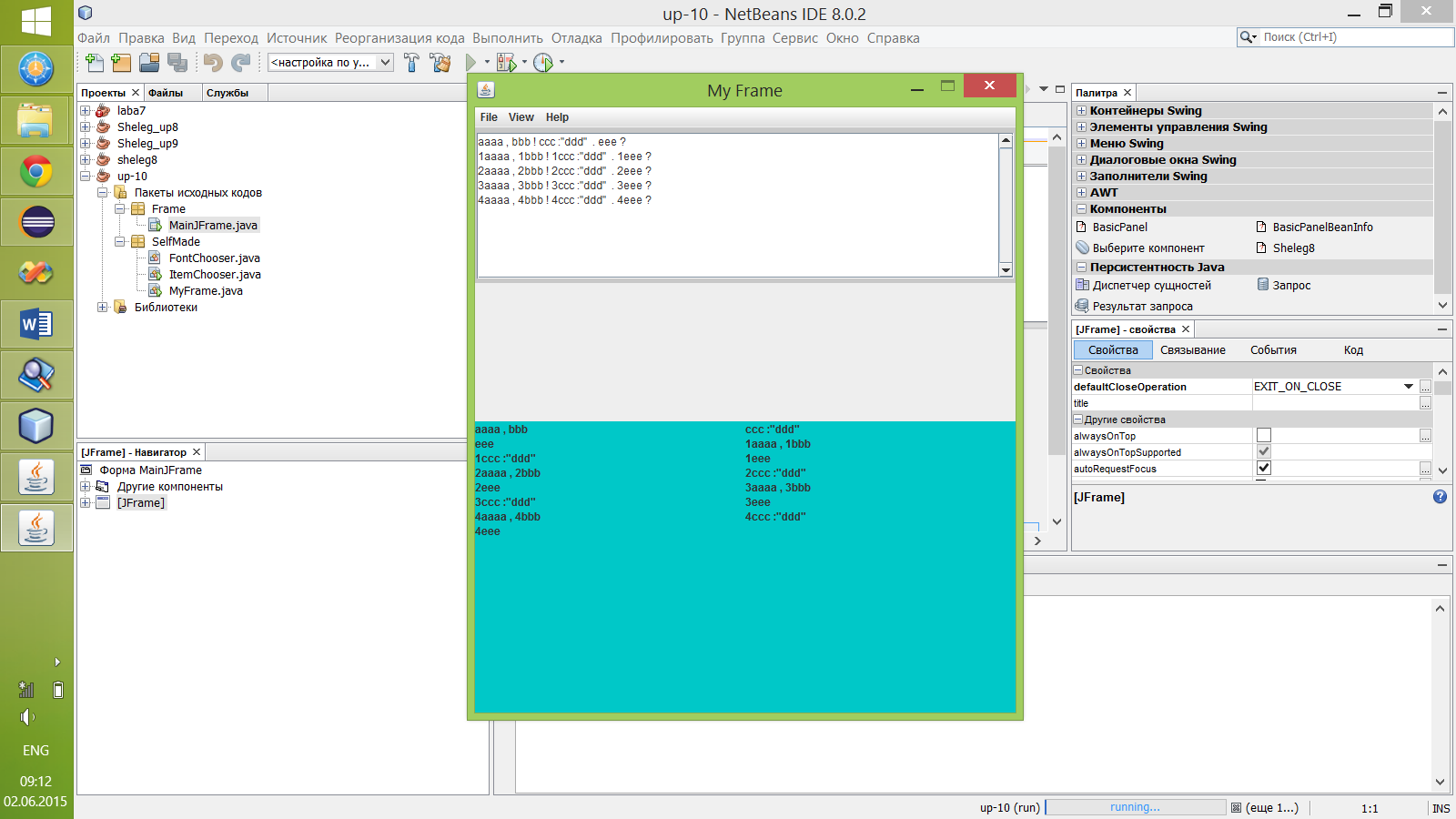
**2.3 Результат:**

Рисунок 10.1 – Результат работы приложения

**3 ВЫВОДЫ**

Для поиска заглавных букв используется метод isUpperCase

Действия после нажатия на кнопку реализуются в обработчике события нажатия. Используется метод actionPerformed

Элементы меню JMenu добавляются в JMenuBar методом add

**11 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №12**

**Цель:**

Научиться использовать нативные методы в Java, когда это необходимо.

**Предметная область:**

Java Native Interface

**1 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

JNI – это механизм, позволяющий вызывать в Java программе функции, написанные на C/++. Причины использования нативных методов могуть быть различными: получение доступа к системным особенностям системы, с которыми легче работать в C языках, произваодительность и др. Важно заметить, что программа с нативными методами не портируема. Портируем лишь Java-часть программы.

**2 ЗАДАНИЕ**

**2.1 Постановка задачи**

Реализовать работу с реестром Windows. Продемонстрировать решение. Для разработки библиотеки использовать Win32 API и MS Visual Studio C/C++.

**2.2 Решение задачи**

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.

**3 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.

**12 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №13**

**Цель:**

Научиться использовать концепцию MUD. Технология RMI.

**Предметная область:**

Сетевые приложения. Удаленный вызов процедур.

**1 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Технологии вызова удаленных методов реализованы в пакетах *java.rmi* и *java.rmi.server*.Вызов удаленных методов является мощной технологией для разработкисетевых 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.

1. 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*.

1. 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.

Обмен сообщениями. Клиент посылает через сервер сообщение другому клиенту, выбранному из списка клиентов, подключенных в данный момент.

**2.2 Решение задачи**

При решении использовалась парадигма MUD и классы из примера №2. Роль почтового сервера играет пользователь mailserver. Для того чтобы отправить письмо своему собеседнику, необходимо «завести разговор с mailserver». После команды talk mailserver приходит ответ для выбора команды. Почтовый сервер предлагает прочитать свою почту либо отправить сообщение какому-либо из пользователей. При этом письмо после прочтения удаляется. Почтовый ящик ограниченного размера. Для запуска не обязательно вручную запускать rmiregistry: нужно просто создать сервер с необходимыми параметрами либо из заранее сохраненного дампа. Учитывая, что реализация из примера сделана для работы в консоли, то и почтовый сервер также был написан консольный.

**2.3 Результат:**

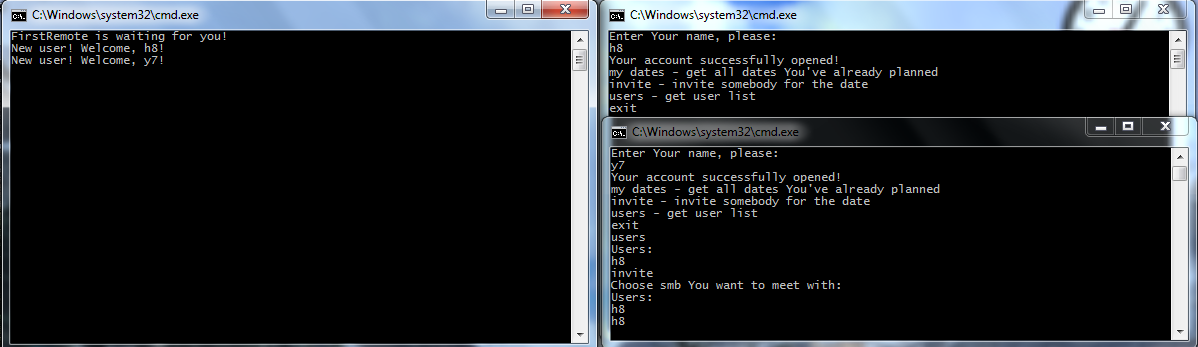


Рисунок 12.1 – Результат работы программы

**3 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*.

1. **ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №14**

**Цель:**

Научиться создавать сервлеты, изучить технологию JSP.

**Предметная область:**

Сервлеты. JSP.

**1 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Сервлеты являются специализированным механизмом Java для создания WEB ресурсов. Сервлеты входят в пакеты javax.servlet, javax.servlet.http, javax.servlet.jsp; пакеты эти, в свою очередь, принадлежат набору Java Servlet API, который входит в архитектуру Java 2 Enterprise Edition. Для работы с сервлетами необходимо иметь J2EE, либо Java Servlet API, поставляемый вместе с такими WEB серверами, как Jacarta Tomcat.

Сервлеты, фактически, это модули обработки HTTP и FTP запросов, используемые для построения порталов (web gates).

Основой этих порталов является собственно WEB сервер - программа, которая держит сокет сервера, принимает и передаёт данные. Чаще всего, для ускорения работы, сервер бывает написан не на Java, а на каком-либо другом языке программирования (например на C++). В связке с сервером работает базовый сервлет. Именно ему отправляет сервер данные от него же получает ответ, отправляемый клиенту. Фактически, базовый сервлет является "мозгом" сервера. Основная функция этого сервлета - прочитать запрос клиента, расшифровать его и, в соответствиии с расшифровкой, передать работу сервлету, отвечающему за этот тип запрашиваемой информации. Зачастую, для достижения скорости, роль базового сервлета играет сам сервер. Именно по такой схеме работает, скажем, Jacarta

Tomcat.

**2 ЗАДАНИЕ**

**2.1 Постановка задачи:**

Изучите части 1-2 книги “Java Servlet & JSP Cookbook”, а также следующие статьи:

Java-УП-14.ppt http://www.javaportal.ru/java/articles/java\_http\_web/article05.html http://www.codenet.ru/webmast/java/jsp.php

Установите на компьютер Apache Tomcat 8.\* и Ant 1.\* Изучить примеры сервлетов TomCat, а также их исходный код.

**2.2 Решение задачи:**

В первую очередь были изучены части 1-2 книги “Java Servlet & JSP Cookbook”, а также следующие статьи:

Java-УП-14.ppt http://www.javaportal.ru/java/articles/java\_http\_web/article05.html http://www.codenet.ru/webmast/java/jsp.php

После этого были установлены Apache Tomcat 8.\* и Ant 1.\*

Были изучены примеры сервлетов TomCat на http://127.0.0.1:8080/examples/servlets/ и http://127.0.0.1:8080/examples/jsp/

**2.3 Результат:**

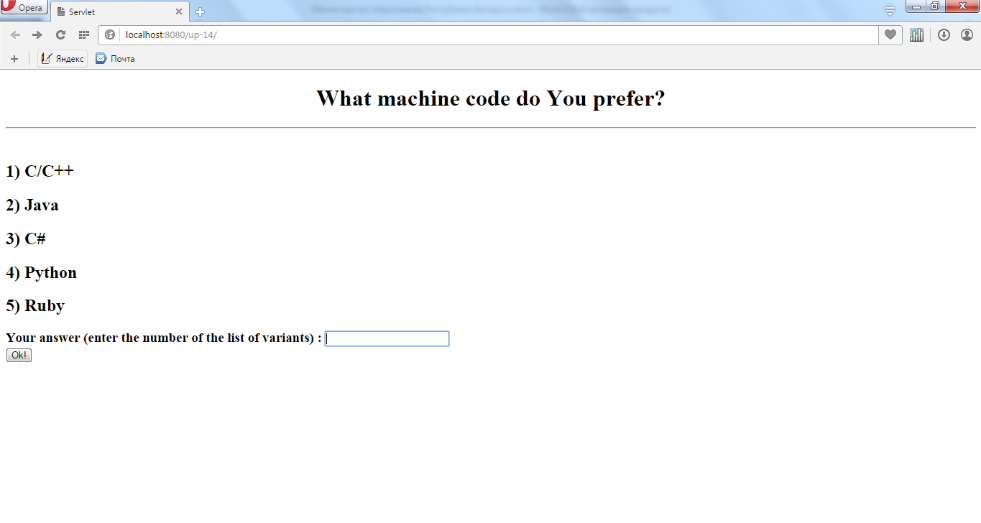


Рисунок 13.1 – пример сервлета

**3 ВЫВОДЫ**

Как правило, сервлет не выполняется один. Он работает в составе Web-приложения. **Web-приложение** (web application) составляют все ресурсы, написанные для обслуживания запросов клиента: сервлеты, JSP, страницы HTML, документы XML, другие документы, изображения и чертежи, музыкальные и видеофайлы. Спецификация "Java Servlet Specification" описывает структуру каталогов, содержащих все эти ресурсы.

Все Web-приложение целиком часто упаковывается в один файл по технологии JAR. Такой файл обычно получает расширение war (Web ARchive). Этот файл можно переносить с одного Web-сервера на другой, при этом многие контейнеры сервлетов могут запускать Web-приложение прямо из архива, не распаковывая его.

**ПРИЛОЖЕНИЕ A**

**Исходные коды лабораторных работ**

**A.1 Исходные коды лабораторной работы №1**

**GraphicsExample.java**

**package sheleg\_up0;**

**import java.awt.\*;**

**/\*\***

**\* This interface defines the methods that must be implemented by an**

**\* object that is to be displayed by the GraphicsExampleFrame object**

**\*/**

**public interface GraphicsExample {**

**public String getTitle(); // Return the example name**

**public int getWidth(); // Return its width**

**public int getHeight(); // Return its height**

**public void draw(Graphics2D g, Component c); // Draw the example**

**}**

**MyShape.java**

**package sheleg\_up0;**

**package sheleg\_up0;**

**import java.awt.Dimension;**

**import java.awt.Point;**

**import java.awt.Rectangle;**

**import java.awt.Shape;**

**import java.awt.geom.AffineTransform;**

**import java.awt.geom.Ellipse2D;**

**import java.awt.geom.PathIterator;**

**import java.awt.geom.Point2D;**

**import java.awt.geom.Rectangle2D;**

**public class MyShape extends Rectangle2D implements Shape**

**{**

**/\*\***

**\* The X coordinate of the top-left corner of the rectangle.**

**\***

**\* @see #setLocation(int, int)**

**\* @see #getLocation()**

**\* @serial the x coordinate**

**\*/**

**public int x;**

**/\*\***

**\* The Y coordinate of the top-left corner of the rectangle.**

**\***

**\* @see #setLocation(int, int)**

**\* @see #getLocation()**

**\* @serial the y coordinate**

**\*/**

**public int y;**

**/\*\***

**\* The width of the rectangle.**

**\***

**\* @see #setSize(int, int)**

**\* @see #getSize()**

**\* @serial**

**\*/**

**public int width;**

**/\*\***

**\* The height of the rectangle.**

**\***

**\* @see #setSize(int, int)**

**\* @see #getSize()**

**\* @serial**

**\*/**

**public int height;**

**/\*\***

**\* Initializes a new instance of <code>Rectangle</code> with a top**

**\* left corner at (0,0) and a width and height of 0.**

**\*/**

**public MyShape()**

**{**

**}**

**/\*\***

**\* Initializes a new instance of <code>Rectangle</code> from the**

**\* coordinates of the specified rectangle.**

**\***

**\* @param r the rectangle to copy from**

**\* @since 1.1**

**\*/**

**public MyShape(Rectangle r)**

**{**

**x = r.x;**

**y = r.y;**

**width = r.width;**

**height = r.height;**

**}**

**/\*\***

**\* Initializes a new instance of <code>Rectangle</code> from the specified**

**\* inputs.**

**\***

**\* @param x the X coordinate of the top left corner**

**\* @param y the Y coordinate of the top left corner**

**\* @param width the width of the rectangle**

**\* @param height the height of the rectangle**

**\*/**

**public MyShape(int x, int y, int width, int height)**

**{**

**this.x = x;**

**this.y = y;**

**this.width = width;**

**this.height = height;**

**}**

**/\*\***

**\* Initializes a new instance of <code>Rectangle</code> with the specified**

**\* width and height. The upper left corner of the rectangle will be at**

**\* the origin (0,0).**

**\***

**\* @param width the width of the rectangle**

**\* @param height the height of the rectange**

**\*/**

**public MyShape(int width, int height)**

**{**

**this.width = width;**

**this.height = height;**

**}**

**/\*\***

**\* Initializes a new instance of <code>Rectangle</code> with a top-left**

**\* corner represented by the specified point and the width and height**

**\* represented by the specified dimension.**

**\***

**\* @param p the upper left corner of the rectangle**

**\* @param d the width and height of the rectangle**

**\*/**

**public MyShape(Point p, Dimension d)**

**{**

**x = p.x;**

**y = p.y;**

**width = d.width;**

**height = d.height;**

**}**

**/\*\***

**\* Initializes a new instance of <code>Rectangle</code> with a top left**

**\* corner at the specified point and a width and height of zero.**

**\***

**\* @param p the upper left corner of the rectangle**

**\*/**

**public MyShape(Point p)**

**{**

**x = p.x;**

**y = p.y;**

**}**

**/\*\***

**\* Initializes a new instance of <code>Rectangle</code> with an**

**\* upper left corner at the origin (0,0) and a width and height represented**

**\* by the specified dimension.**

**\***

**\* @param d the width and height of the rectangle**

**\*/**

**public MyShape(Dimension d)**

**{**

**width = d.width;**

**height = d.height;**

**}**

**/\*\***

**\* Get the X coordinate of the upper-left corner.**

**\***

**\* @return the value of x, as a double**

**\*/**

**public double getX()**

**{**

**return x;**

**}**

**/\*\***

**\* Get the Y coordinate of the upper-left corner.**

**\***

**\* @return the value of y, as a double**

**\*/**

**public double getY()**

**{**

**return y;**

**}**

**/\*\***

**\* Get the width of the rectangle.**

**\***

**\* @return the value of width, as a double**

**\*/**

**public double getWidth()**

**{**

**return width;**

**}**

**/\*\***

**\* Get the height of the rectangle.**

**\***

**\* @return the value of height, as a double**

**\*/**

**public double getHeight()**

**{**

**return height;**

**}**

**/\*\***

**\* Returns the bounds of this rectangle. A pretty useless method, as this**

**\* is already a rectangle; it is included to mimic the**

**\* <code>getBounds</code> method in Component.**

**\***

**\* @return a copy of this rectangle**

**\* @see #setBounds(Rectangle)**

**\* @since 1.1**

**\*/**

**public Rectangle getBounds()**

**{**

**return new Rectangle(this.x,this.y,this.width,this.height);**

**}**

**/\*\***

**\* Returns the high-precision bounds of this rectangle. A pretty useless**

**\* method, as this is already a rectangle.**

**\***

**\* @return a copy of this rectangle**

**\* @see #setBounds(Rectangle)**

**\* @since 1.2**

**\*/**

**public Rectangle2D getBounds2D()**

**{**

**return new Rectangle(x, y, width, height);**

**}**

**/\*\***

**\* Updates this rectangle to match the dimensions of the specified**

**\* rectangle.**

**\***

**\* @param r the rectangle to update from**

**\* @throws NullPointerException if r is null**

**\* @see #setBounds(int, int, int, int)**

**\* @since 1.1**

**\*/**

**public void setBounds(Rectangle r)**

**{**

**setBounds (r.x, r.y, r.width, r.height);**

**}**

**/\*\***

**\* Updates this rectangle to have the specified dimensions.**

**\***

**\* @param x the new X coordinate of the upper left hand corner**

**\* @param y the new Y coordinate of the upper left hand corner**

**\* @param width the new width of this rectangle**

**\* @param height the new height of this rectangle**

**\* @since 1.1**

**\*/**

**public void setBounds(int x, int y, int width, int height)**

**{**

**reshape (x, y, width, height);**

**}**

**/\*\***

**\* Updates this rectangle to have the specified dimensions, rounded to the**

**\* integer precision used by this class (the values are rounded "outwards" so**

**\* that the stored rectangle completely encloses the specified double**

**\* precision rectangle).**

**\***

**\* @param x the new X coordinate of the upper left hand corner**

**\* @param y the new Y coordinate of the upper left hand corner**

**\* @param width the new width of this rectangle**

**\* @param height the new height of this rectangle**

**\* @since 1.2**

**\*/**

**public void setRect(double x, double y, double width, double height)**

**{**

**this.x = (int) Math.floor(x);**

**this.y = (int) Math.floor(y);**

**this.width = (int) Math.ceil(x + width) - this.x;**

**this.height = (int) Math.ceil(y + height) - this.y;**

**}**

**/\*\***

**\* Updates this rectangle to have the specified dimensions.**

**\***

**\* @param x the new X coordinate of the upper left hand corner**

**\* @param y the new Y coordinate of the upper left hand corner**

**\* @param width the new width of this rectangle**

**\* @param height the new height of this rectangle**

**\* @deprecated use {@link #setBounds(int, int, int, int)} instead**

**\*/**

**public void reshape(int x, int y, int width, int height)**

**{**

**this.x = x;**

**this.y = y;**

**this.width = width;**

**this.height = height;**

**}**

**/\*\***

**\* Returns the location of this rectangle, which is the coordinates of**

**\* its upper left corner.**

**\***

**\* @return the point where this rectangle is located**

**\* @see #setLocation(Point)**

**\* @since 1.1**

**\*/**

**public Point getLocation()**

**{**

**return new Point(x,y);**

**}**

**/\*\***

**\* Moves the location of this rectangle by setting its upper left**

**\* corner to the specified point.**

**\***

**\* @param p the point to move the rectangle to**

**\* @throws NullPointerException if p is null**

**\* @see #getLocation()**

**\* @since 1.1**

**\*/**

**public void setLocation(Point p)**

**{**

**setLocation (p.x, p.y);**

**}**

**/\*\***

**\* Moves the location of this rectangle by setting its upper left**

**\* corner to the specified coordinates.**

**\***

**\* @param x the new X coordinate for this rectangle**

**\* @param y the new Y coordinate for this rectangle**

**\* @since 1.1**

**\*/**

**public void setLocation(int x, int y)**

**{**

**move (x, y);**

**}**

**/\*\***

**\* Moves the location of this rectangle by setting its upper left**

**\* corner to the specified coordinates.**

**\***

**\* @param x the new X coordinate for this rectangle**

**\* @param y the new Y coordinate for this rectangle**

**\* @deprecated use {@link #setLocation(int, int)} instead**

**\*/**

**public void move(int x, int y)**

**{**

**this.x = x;**

**this.y = y;**

**}**

**/\*\***

**\* Translate the location of this rectangle by the given amounts.**

**\***

**\* @param dx the x distance to move by**

**\* @param dy the y distance to move by**

**\* @see #setLocation(int, int)**

**\*/**

**public void translate(int dx, int dy)**

**{**

**x += dx;**

**y += dy;**

**}**

**/\*\***

**\* Returns the size of this rectangle.**

**\***

**\* @return the size of this rectangle**

**\* @see #setSize(Dimension)**

**\* @since 1.1**

**\*/**

**public Dimension getSize()**

**{**

**return new Dimension(width, height);**

**}**

**/\*\***

**\* Sets the size of this rectangle based on the specified dimensions.**

**\***

**\* @param d the new dimensions of the rectangle**

**\* @throws NullPointerException if d is null**

**\* @see #getSize()**

**\* @since 1.1**

**\*/**

**public void setSize(Dimension d)**

**{**

**setSize (d.width, d.height);**

**}**

**/\*\***

**\* Sets the size of this rectangle based on the specified dimensions.**

**\***

**\* @param width the new width of the rectangle**

**\* @param height the new height of the rectangle**

**\* @since 1.1**

**\*/**

**public void setSize(int width, int height)**

**{**

**resize (width, height);**

**}**

**/\*\***

**\* Sets the size of this rectangle based on the specified dimensions.**

**\***

**\* @param width the new width of the rectangle**

**\* @param height the new height of the rectangle**

**\* @deprecated use {@link #setSize(int, int)} instead**

**\*/**

**public void resize(int width, int height)**

**{**

**this.width = width;**

**this.height = height;**

**}**

**/\*\***

**\* Tests whether or not the specified point is inside this rectangle.**

**\* According to the contract of Shape, a point on the border is in only if**

**\* it has an adjacent point inside the rectangle in either the increasing**

**\* x or y direction.**

**\***

**\* @param p the point to test**

**\* @return true if the point is inside the rectangle**

**\* @throws NullPointerException if p is null**

**\* @see #contains(int, int)**

**\* @since 1.1**

**\*/**

**public boolean contains(Point p)**

**{**

**return contains (p.x, p.y);**

**}**

**/\*\***

**\* Tests whether or not the specified point is inside this rectangle.**

**\* According to the contract of Shape, a point on the border is in only if**

**\* it has an adjacent point inside the rectangle in either the increasing**

**\* x or y direction.**

**\***

**\* @param x the X coordinate of the point to test**

**\* @param y the Y coordinate of the point to test**

**\* @return true if the point is inside the rectangle**

**\* @since 1.1**

**\*/**

**public boolean contains(int x, int y)**

**{**

**return inside (x, y);**

**}**

**/\*\***

**\* Checks whether all points in the given rectangle are contained in this**

**\* rectangle.**

**\***

**\* @param r the rectangle to check**

**\* @return true if r is contained in this rectangle**

**\* @throws NullPointerException if r is null**

**\* @see #contains(int, int, int, int)**

**\* @since 1.1**

**\*/**

**public boolean contains(Rectangle r)**

**{**

**return contains (r.x, r.y, r.width, r.height);**

**}**

**/\*\***

**\* Checks whether all points in the given rectangle are contained in this**

**\* rectangle.**

**\***

**\* @param x the x coordinate of the rectangle to check**

**\* @param y the y coordinate of the rectangle to check**

**\* @param w the width of the rectangle to check**

**\* @param h the height of the rectangle to check**

**\* @return true if the parameters are contained in this rectangle**

**\* @since 1.1**

**\*/**

**public boolean contains(int x, int y, int w, int h)**

**{**

**return width > 0 && height > 0 && w > 0 && h > 0**

**&& x >= this.x && x + w <= this.x + this.width**

**&& y >= this.y && y + h <= this.y + this.height;**

**}**

**/\*\***

**\* Tests whether or not the specified point is inside this rectangle.**

**\***

**\* @param x the X coordinate of the point to test**

**\* @param y the Y coordinate of the point to test**

**\* @return true if the point is inside the rectangle**

**\* @deprecated use {@link #contains(int, int)} instead**

**\*/**

**public boolean inside(int x, int y)**

**{**

**return width > 0 && height > 0**

**&& x >= this.x && x < this.x + width**

**&& y >= this.y && y < this.y + height;**

**}**

**/\*\***

**\* Tests whether or not the specified rectangle intersects this rectangle.**

**\* This means the two rectangles share at least one internal point.**

**\***

**\* @param r the rectangle to test against**

**\* @return true if the specified rectangle intersects this one**

**\* @throws NullPointerException if r is null**

**\* @since 1.2**

**\*/**

**public boolean intersects(Rectangle r)**

**{**

**return r.width > 0 && r.height > 0 && width > 0 && height > 0**

**&& r.x < x + width && r.x + r.width > x**

**&& r.y < y + height && r.y + r.height > y;**

**}**

**/\*\***

**\* Determines the rectangle which is formed by the intersection of this**

**\* rectangle with the specified rectangle. If the two do not intersect,**

**\* an empty rectangle will be returned (meaning the width and/or height**

**\* will be non-positive).**

**\***

**\* @param r the rectange to calculate the intersection with**

**\* @return a new rectangle bounding the intersection**

**\* @throws NullPointerException if r is null**

**\*/**

**public Rectangle intersection(Rectangle r)**

**{**

**Rectangle res = new Rectangle();**

**intersect(this, r, res);**

**return res;**

**}**

**/\*\***

**\* Returns the smallest rectangle that contains both this rectangle**

**\* and the specified rectangle.**

**\***

**\* @param r the rectangle to compute the union with**

**\* @return the smallest rectangle containing both rectangles**

**\* @throws NullPointerException if r is null**

**\*/**

**public Rectangle union(Rectangle r)**

**{**

**Rectangle res = new Rectangle();**

**union(this, r, res);**

**return res;**

**}**

**/\*\***

**\* Modifies this rectangle so that it represents the smallest rectangle**

**\* that contains both the existing rectangle and the specified point.**

**\* However, if the point falls on one of the two borders which are not**

**\* inside the rectangle, a subsequent call to <code>contains</code> may**

**\* return false.**

**\***

**\* @param x the X coordinate of the point to add to this rectangle**

**\* @param y the Y coordinate of the point to add to this rectangle**

**\*/**

**public void add(int x, int y)**

**{**

**add((double) x, (double) y);**

**}**

**/\*\***

**\* Modifies this rectangle so that it represents the smallest rectangle**

**\* that contains both the existing rectangle and the specified point.**

**\* However, if the point falls on one of the two borders which are not**

**\* inside the rectangle, a subsequent call to <code>contains</code> may**

**\* return false.**

**\***

**\* @param p the point to add to this rectangle**

**\* @throws NullPointerException if p is null**

**\*/**

**public void add(Point p)**

**{**

**add((double) p.x, (double) p.y);**

**}**

**/\*\***

**\* Modifies this rectangle so that it represents the smallest rectangle**

**\* that contains both the existing rectangle and the specified rectangle.**

**\***

**\* @param r the rectangle to add to this rectangle**

**\* @throws NullPointerException if r is null**

**\* @see #union(Rectangle)**

**\*/**

**public void add(Rectangle r)**

**{**

**union(this, r, this);**

**}**

**/\*\***

**\* Expands the rectangle by the specified amount. The horizontal**

**\* and vertical expansion values are applied both to the X,Y coordinate**

**\* of this rectangle, and its width and height. Thus the width and**

**\* height will increase by 2h and 2v accordingly.**

**\***

**\* @param h the horizontal expansion value**

**\* @param v the vertical expansion value**

**\*/**

**public void grow(int h, int v)**

**{**

**x -= h;**

**y -= v;**

**width += h + h;**

**height += v + v;**

**}**

**/\*\***

**\* Tests whether or not this rectangle is empty. An empty rectangle**

**\* has a non-positive width or height.**

**\***

**\* @return true if the rectangle is empty**

**\*/**

**public boolean isEmpty()**

**{**

**return width <= 0 || height <= 0;**

**}**

**/\*\***

**\* Determine where the point lies with respect to this rectangle. The**

**\* result will be the binary OR of the appropriate bit masks.**

**\***

**\* @param x the x coordinate to check**

**\* @param y the y coordinate to check**

**\* @return the binary OR of the result**

**\* @see #OUT\_LEFT**

**\* @see #OUT\_TOP**

**\* @see #OUT\_RIGHT**

**\* @see #OUT\_BOTTOM**

**\* @since 1.2**

**\*/**

**public int outcode(double x, double y)**

**{**

**int result = 0;**

**if (width <= 0)**

**result |= OUT\_LEFT | OUT\_RIGHT;**

**else if (x < this.x)**

**result |= OUT\_LEFT;**

**else if (x > this.x + width)**

**result |= OUT\_RIGHT;**

**if (height <= 0)**

**result |= OUT\_BOTTOM | OUT\_TOP;**

**else if (y < this.y) // Remember that +y heads top-to-bottom.**

**result |= OUT\_TOP;**

**else if (y > this.y + height)**

**result |= OUT\_BOTTOM;**

**return result;**

**}**

**/\*\***

**\* Determines the rectangle which is formed by the intersection of this**

**\* rectangle with the specified rectangle. If the two do not intersect,**

**\* an empty rectangle will be returned (meaning the width and/or height**

**\* will be non-positive).**

**\***

**\* @param r the rectange to calculate the intersection with**

**\* @return a new rectangle bounding the intersection**

**\* @throws NullPointerException if r is null**

**\* @since 1.2**

**\*/**

**public Rectangle2D createIntersection(Rectangle2D r)**

**{**

**// Favor runtime type of other rectangle.**

**Rectangle2D res = r.getBounds2D();**

**intersect(this, r, res);**

**return res;**

**}**

**/\*\***

**\* Returns the smallest rectangle that contains both this rectangle**

**\* and the specified rectangle.**

**\***

**\* @param r the rectangle to compute the union with**

**\* @return the smallest rectangle containing both rectangles**

**\* @throws NullPointerException if r is null**

**\* @since 1.2**

**\*/**

**public Rectangle2D createUnion(Rectangle2D r)**

**{**

**// Favor runtime type of other rectangle.**

**Rectangle2D res = r.getBounds2D();**

**union(this, r, res);**

**return res;**

**}**

**/\*\***

**\* Tests this rectangle for equality against the specified object. This**

**\* will be true if an only if the specified object is an instance of**

**\* Rectangle2D with the same coordinates and dimensions.**

**\***

**\* @param obj the object to test against for equality**

**\* @return true if the specified object is equal to this one**

**\*/**

**public boolean equals(Object obj)**

**{**

**// NOTE: No special hashCode() method is required for this class,**

**// as this equals() implementation is functionally equivalent to**

**// super.equals(), which does define a proper hashCode().**

**if (! (obj instanceof Rectangle2D))**

**return false;**

**Rectangle2D r = (Rectangle2D) obj;**

**return r.getX() == x && r.getY() == y**

**&& r.getWidth() == width && r.getHeight() == height;**

**}**

**/\*\***

**\* Returns a string representation of this rectangle. This is in the form**

**\* <code>getClass().getName() + "[x=" + x + ",y=" + y + ",width=" + width**

**\* + ",height=" + height + ']'</code>.**

**\***

**\* @return a string representation of this rectangle**

**\*/**

**public String toString()**

**{**

**return getClass().getName() + "[x=" + x + ",y=" + y + ",width=" + width**

**+ ",height=" + height + ']';**

**}**

**}**

**Transforms.java**

**package sheleg\_up0;**

**import java.awt.\*;**

**import java.awt.event.\*;**

**import javax.swing.\*;**

**import javax.swing.event.\*;**

**import java.awt.print.\*;**

**import java.awt.geom.\*;**

**/\*\* A demonstration of Java2D transformations \*/**

**public class Transforms extends Thread implements GraphicsExample {**

**public String getTitle() {**

**return "Rectangle and Ellipse";**

**} // From GraphicsExample**

**public int getWidth() {**

**return 400;**

**} // From GraphicsExample**

**public int getHeight() {**

**return 400;**

**} // From GraphicsExample**

**protected double angle=0;**

**protected Component parent=null;**

**protected Shape ellipse;**

**protected Shape rectangle;**

**protected AffineTransform transform;**

**private double width;**

**private Color bcolor;**

**private Color fcolor;**

**/\*\***

**\* This constructor sets up the Shape and AffineTransform objects we need**

**\*\*/**

**public Transforms() {**

**ellipse=new Ellipse2D.Float(0, 0, 200, 120);**

**//rectangle=new Rectangle2D.Float(0, 0, 200, 120);**

**rectangle=new MyShape(0, 0, 200, 120);**

**transform=AffineTransform.getRotateInstance(0,0,0);**

**width=3;**

**bcolor=Color.black;**

**fcolor=Color.white;**

**if(GraphicsExampleFrame.args.length==3)**

**{**

**width=GraphicsExampleFrame.getDouble(GraphicsExampleFrame.args[0],width);**

**bcolor=GraphicsExampleFrame.getHtmlColor(GraphicsExampleFrame.args[1],bcolor);**

**fcolor=GraphicsExampleFrame.getHtmlColor(GraphicsExampleFrame.args[2],fcolor);**

**}**

**}**

**/\*\* Draw the defined shape and label, using each transform \*/**

**public void draw(Graphics2D g, Component c) {**

**parent=c;**

**// Define basic drawing attributes**

**g.setStroke(new BasicStroke((float)width, BasicStroke.CAP\_SQUARE, // 2-pixel**

**BasicStroke.JOIN\_BEVEL));**

**g.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, // antialias**

**RenderingHints.VALUE\_ANTIALIAS\_ON);**

**AffineTransform save = g.getTransform();**

**/\*g.translate(100, 100);**

**transform.setToRotation(-angle,100,60);**

**g.transform(transform); // apply transform**

**g.setColor(fcolor);**

**g.fill(rectangle);**

**g.setColor(bcolor);**

**g.draw(rectangle);**

**g.setTransform(save);\*/**

**g.translate(100, 100);**

**transform.setToRotation(-angle,100,60);**

**g.transform(transform); // apply transform**

**g.setColor(fcolor);**

**g.fill(ellipse);**

**g.setColor(bcolor);**

**g.draw(ellipse);**

**g.setTransform(save);**

**AffineTransform.getRotateInstance(angle);**

**}**

**public void run() {**

**while ( true ) {**

**try {**

**Thread.sleep(100);**

**} catch ( Exception e ) {**

**break;**

**}**

**if(isInterrupted()==false){**

**angle+=0.1;**

**if(angle>=Math.PI\*2)**

**angle=0;**

**if(parent!=null)**

**parent.repaint();**

**}**

**}**

**}**

**}**

**// //////////////////////////////////////////////////////////////////////////**

**// Frame**

**// //////////////////////////////////////////////////////////////////////////**

**class GraphicsExampleFrame extends JFrame {**

**// The class name of the requested example**

**static final String classname = "sheleg\_up0.Transforms";**

**public GraphicsExampleFrame(final GraphicsExample[] examples) {**

**super("GraphicsExampleFrame");**

**Container cpane = getContentPane(); // Set up the frame**

**cpane.setLayout(new BorderLayout());**

**final JTabbedPane tpane = new JTabbedPane(); // And the tabbed pane**

**cpane.add(tpane, BorderLayout.CENTER);**

**// Add a menubar**

**JMenuBar menubar = new JMenuBar(); // Create the menubar**

**this.setJMenuBar(menubar); // Add it to the frame**

**JMenu filemenu = new JMenu("File"); // Create a File menu**

**menubar.add(filemenu); // Add to the menubar**

**JMenuItem print = new JMenuItem("Print"); // Create a Print item**

**filemenu.add(print); // Add it to the menu**

**JMenuItem quit = new JMenuItem("Quit"); // Create a Quit item**

**filemenu.add(quit); // Add it to the menu**

**// Tell the Print menu item what to do when selected**

**print.addActionListener(new ActionListener() {**

**public void actionPerformed(ActionEvent e) {**

**// Get the currently displayed example, and call**

**// the print method (defined below)**

**print(examples[tpane.getSelectedIndex()]);**

**}**

**});**

**// Tell the Quit menu item what to do when selected**

**quit.addActionListener(new ActionListener() {**

**public void actionPerformed(ActionEvent e) {**

**System.exit(0);**

**}**

**});**

**// In addition to the Quit menu item, also handle window close events**

**this.addWindowListener(new WindowAdapter() {**

**public void windowClosing(WindowEvent e) {**

**System.exit(0);**

**}**

**});**

**// Insert each of the example objects into the tabbed pane**

**for (int i = 0; i < examples.length; i++) {**

**GraphicsExample e = examples[i];**

**tpane.addTab(e.getTitle(), new GraphicsExamplePane(e));**

**}**

**}**

**/\*\***

**\* This inner class is a custom Swing component that displays a**

**\* GraphicsExample object.**

**\*/**

**public class GraphicsExamplePane extends JComponent {**

**GraphicsExample example; // The example to display**

**Dimension size; // How much space it requires**

**public GraphicsExamplePane(GraphicsExample example) {**

**this.example = example;**

**size = new Dimension(example.getWidth(), example.getHeight());**

**}**

**/\*\* Draw the component and the example it contains \*/**

**public void paintComponent(Graphics g) {**

**g.setColor(Color.white); // set the background**

**g.fillRect(0, 0, size.width, size.height); // to white**

**g.setColor(Color.red); // set a default drawing color**

**example.draw((Graphics2D) g, this); // ask example to draw itself**

**}**

**// These methods specify how big the component must be**

**public Dimension getPreferredSize() {**

**return size;**

**}**

**public Dimension getMinimumSize() {**

**return size;**

**}**

**}**

**/\*\* This method is invoked by the Print menu item \*/**

**public void print(final GraphicsExample example) {**

**// Start off by getting a printer job to do the printing**

**PrinterJob job = PrinterJob.getPrinterJob();**

**// Wrap the example in a Printable object (defined below)**

**// and tell the PrinterJob that we want to print it**

**job.setPrintable(new PrintableExample(example));**

**// Display the print dialog to the user**

**if (job.printDialog()) {**

**// If they didn't cancel it, then tell the job to start printing**

**try {**

**job.print();**

**} catch (PrinterException e) {**

**System.out.println("Couldn't print: " + e.getMessage());**

**}**

**}**

**}**

**/\*\***

**\* This inner class implements the Printable interface in order to print a**

**\* GraphicsExample object.**

**\*\*/**

**class PrintableExample implements Printable {**

**GraphicsExample example; // The example to print**

**// The constructor. Just remember the example**

**public PrintableExample(GraphicsExample example) {**

**this.example = example;**

**}**

**/\*\***

**\* This method is called by the PrinterJob to print the example**

**\*\*/**

**public int print(Graphics g, PageFormat pf, int pageIndex) {**

**// Tell the PrinterJob that there is only one page**

**if (pageIndex != 0)**

**return NO\_SUCH\_PAGE;**

**// The PrinterJob supplies us a Graphics object to draw with.**

**// Anything drawn with this object will be sent to the printer.**

**// The Graphics object can safely be cast to a Graphics2D object.**

**Graphics2D g2 = (Graphics2D) g;**

**// Translate to skip the left and top margins.**

**g2.translate(pf.getImageableX(), pf.getImageableY());**

**// Figure out how big the printable area is, and how big**

**// the example is.**

**double pageWidth = pf.getImageableWidth();**

**double pageHeight = pf.getImageableHeight();**

**double exampleWidth = example.getWidth();**

**double exampleHeight = example.getHeight();**

**// Scale the example if needed**

**double scalex = 1.0, scaley = 1.0;**

**if (exampleWidth > pageWidth)**

**scalex = pageWidth / exampleWidth;**

**if (exampleHeight > pageHeight)**

**scaley = pageHeight / exampleHeight;**

**double scalefactor = Math.min(scalex, scaley);**

**if (scalefactor != 1)**

**g2.scale(scalefactor, scalefactor);**

**// Finally, call the draw() method of the example, passing in**

**// the Graphics2D object for the printer**

**example.draw(g2, GraphicsExampleFrame.this);**

**// Tell the PrinterJob that we successfully printed the page**

**return PAGE\_EXISTS;**

**}**

**}**

**public static String[] args;**

**public static void main(String[] args) {**

**GraphicsExampleFrame.args=args;**

**GraphicsExample[] examples = new GraphicsExample[1];**

**// Try to instantiate the named GraphicsExample class**

**try {**

**Class exampleClass = Class.forName(classname);**

**examples[0] = (GraphicsExample) exampleClass.newInstance();**

**Thread t=(Thread)examples[0];**

**t.start();**

**} catch (ClassNotFoundException e) { // unknown class**

**System.err.println("Couldn't find example: " + classname);**

**System.exit(1);**

**} catch (ClassCastException e) { // wrong type of class**

**System.err.println("Class " + classname**

**+ " is not a GraphicsExample");**

**System.exit(1);**

**} catch (Exception e) { // class doesn't have a public constructor**

**// catch InstantiationException, IllegalAccessException**

**System.err.println("Couldn't instantiate example: " + classname);**

**System.exit(1);**

**}**

**// Now create a window to display the examples in, and make it visible**

**GraphicsExampleFrame f = new GraphicsExampleFrame(examples);**

**f.pack();**

**f.setVisible(true);**

**}**

**public static Color getHtmlColor( String strRGB, Color def ) {**

**if ( strRGB != null && strRGB.charAt(0)== '#' ) {**

**try {**

**return new Color(**

**Integer.parseInt( strRGB.substring( 1 ), 16 ) );**

**} catch ( NumberFormatException e ) {**

**return def;**

**}**

**}**

**return def;**

**}**

**public static double getDouble( String str, double def ) {**

**if ( str != null ) {**

**try {**

**return Double.parseDouble(str);**

**} catch ( NumberFormatException e ) {**

**return def;**

**}**

**}**

**return def;**

**}**

**}**

**A.2 Исходные коды лабораторной работы №2**

**GenericPaint.java**

**package sheleg\_up1;**

**import java.awt.\*;**

**import java.awt.geom.\*;**

**import java.awt.image.\*;**

**/\*\***

**\* This is an abstract Paint implementation that computes the color of each**

**\* point to be painted by passing the coordinates of the point to the calling**

**\* the abstract methods computeRed(), computeGreen(), computeBlue() and**

**\* computeAlpha(). Subclasses must implement these three methods to perform**

**\* whatever type of painting is desired. Note that while this class provides**

**\* great flexibility, it is not very efficient.**

**\*\*/**

**public abstract class GenericPaint implements Paint {**

**/\*\* This is the main Paint method; all it does is return a PaintContext \*/**

**public PaintContext createContext(ColorModel cm,**

**Rectangle deviceBounds,**

**Rectangle2D userBounds,**

**AffineTransform xform,**

**RenderingHints hints) {**

**return new GenericPaintContext(xform);**

**}**

**/\*\* This paint class allows translucent painting \*/**

**public int getTransparency() { return TRANSLUCENT; }**

**/\*\***

**\* These three methods return the red, green, blue, and alpha values of**

**\* the pixel at appear at the specified user-space coordinates. The return**

**\* value of each method should be between 0 and 255.**

**\*\*/**

**public abstract int computeRed(double x, double y);**

**public abstract int computeGreen(double x, double y);**

**public abstract int computeBlue(double x, double y);**

**public abstract int computeAlpha(double x, double y);**

**/\*\***

**\* The PaintContext class does all the work of painting**

**\*\*/**

**class GenericPaintContext implements PaintContext {**

**ColorModel model; // The color model**

**Point2D origin, unitVectorX, unitVectorY; // For device-to-user xform**

**public GenericPaintContext(AffineTransform userToDevice) {**

**// Our color model packs ARGB values into a single int**

**model = new DirectColorModel(32, 0x00ff0000,0x0000ff00,**

**0x000000ff, 0xff000000);**

**// The specified transform converts user to device pixels**

**// We need to figure out the reverse transformation, so we**

**// can compute the user space coordinates of each device pixel**

**try {**

**AffineTransform deviceToUser = userToDevice.createInverse();**

**origin = deviceToUser.transform(new Point(0,0), null);**

**unitVectorX = deviceToUser.deltaTransform(new Point(1,0),null);**

**unitVectorY = deviceToUser.deltaTransform(new Point(0,1),null);**

**}**

**catch (NoninvertibleTransformException e) {**

**// If we can't invert the transform, just use device space**

**origin = new Point(0,0);**

**unitVectorX = new Point(1,0);**

**unitVectorY = new Point(0, 1);**

**}**

**}**

**/\*\* Return the color model used by this Paint implementation \*/**

**public ColorModel getColorModel() { return model; }**

**/\*\***

**\* This is the main method of PaintContext. It must return a Raster**

**\* that contains fill data for the specified rectangle. It creates a**

**\* raster of the specified size, and loops through the device pixels.**

**\* For each one, it converts the coordinates to user space, then calls**

**\* the computeRed(), computeGreen() and computeBlue() methods to**

**\* obtain the appropriate color for the device pixel.**

**\*\*/**

**public Raster getRaster(int x, int y, int w, int h) {**

**WritableRaster raster = model.createCompatibleWritableRaster(w,h);**

**int[] colorComponents = new int[4];**

**for(int j = 0; j < h; j++) { // Loop through rows of raster**

**int deviceY = y + j;**

**for(int i = 0; i < w; i++) { // Loop through columns**

**int deviceX = x + i;**

**// Convert device coordinate to user-space coordinate**

**double userX = origin.getX() +**

**deviceX \* unitVectorX.getX() +**

**deviceY \* unitVectorY.getX();**

**double userY = origin.getY() +**

**deviceX \* unitVectorX.getY() +**

**deviceY \* unitVectorY.getY();**

**// Compute the color components of the pixel**

**colorComponents[0] = computeRed(userX, userY);**

**colorComponents[1] = computeGreen(userX, userY);**

**colorComponents[2] = computeBlue(userX, userY);**

**colorComponents[3] = computeAlpha(userX, userY);**

**// Set the color of the pixel**

**raster.setPixel(i, j, colorComponents);**

**}**

**}**

**return raster;**

**}**

**/\*\* Called when the PaintContext is no longer needed. \*/**

**public void dispose() {}**

**}**

**}**

**Paints.java**

**package sheleg\_up1;**

**import java.awt.\*;**

**import java.awt.event.\*;**

**import javax.swing.\*;**

**import javax.swing.event.\*;**

**import java.awt.geom.\*;**

**import java.awt.font.\*;**

**import java.awt.image.\*;**

**/\*\* A demonstration of Java2D transformations \*/**

**public class Paints implements GraphSample {**

**static final int WIDTH = 700, HEIGHT = 370; // Size of our example**

**public String getName() { return "Paints"; } // From GraphSample**

**public int getWidth() { return WIDTH; } // From GraphSample**

**public int getHeight() { return HEIGHT; } // From GraphSample**

**/\*\* Draw the example \*/**

**public void draw(Graphics2D ga, Component c) {**

**int width=300;**

**int height=200;**

**BufferedImage image = new BufferedImage(width, height, BufferedImage.TYPE\_INT\_RGB);**

**Graphics2D g = image.createGraphics();**

**g.setRenderingHint(RenderingHints.KEY\_ANTIALIASING,RenderingHints.VALUE\_ANTIALIAS\_ON);**

**g.setPaint(new GradientPaint(0,0,Color.decode("#C9C9C9"),**

**width, 0, Color.decode("#787878")));**

**g.fillRect(0, 0, width, height);**

**Color foreground=Color.decode("#FF0000");**

**g.setPaint(foreground);**

**g.setStroke(new BasicStroke(16));**

**g.drawRect(7, 7, width-15,height-15);**

**Font font = new Font("Serif", Font.BOLD, 8);**

**Font bigfont = font.deriveFont(AffineTransform.getScaleInstance(8.0, 8.0));**

**GlyphVector gv = bigfont.createGlyphVector(g.getFontRenderContext(),"STOP");**

**Paint shadowPaint = new Color(0, 0, 0, 100);**

**AffineTransform shadowTransform = AffineTransform.getShearInstance(-1.0, 0.0);**

**shadowTransform.scale(1.0, 0.5);**

**g.setStroke(new BasicStroke(5.0f));**

**g.translate(60, 120);**

**g.setComposite(AlphaComposite.SrcIn);**

**for(int i=0;i<gv.getNumGlyphs();i++)**

**{**

**g.setPaint(shadowPaint);**

**g.translate(3,4);**

**g.fill(shadowTransform.createTransformedShape(gv.getGlyphOutline(i)));**

**g.translate(-3,-4);**

**g.setPaint(foreground);**

**g.fill(gv.getGlyphOutline(i));**

**}**

**ga.translate(25,80);**

**ga.drawImage(image, AffineTransform.getRotateInstance(Math.PI/4,width/2,height/2), c);**

**ga.drawImage(image, 360,0, c);**

**}**

**}**

**////////////////////////////////////////////////////////////////////////////**

**// Frame**

**////////////////////////////////////////////////////////////////////////////**

**class GraphSampleFrame extends JFrame {**

**// The class name of the requested example**

**static final String classname = "sheleg\_up1.Paints";**

**public GraphSampleFrame(final GraphSample[] examples) {**

**super("GraphSampleFrame");**

**Container cpane = getContentPane(); // Set up the frame**

**cpane.setLayout(new BorderLayout());**

**final JTabbedPane tpane = new JTabbedPane(); // And the tabbed pane**

**cpane.add(tpane, BorderLayout.CENTER);**

**// Add a menubar**

**JMenuBar menubar = new JMenuBar(); // Create the menubar**

**this.setJMenuBar(menubar); // Add it to the frame**

**JMenu filemenu = new JMenu("File"); // Create a File menu**

**menubar.add(filemenu); // Add to the menubar**

**JMenuItem quit = new JMenuItem("Quit"); // Create a Quit item**

**filemenu.add(quit); // Add it to the menu**

**// Tell the Quit menu item what to do when selected**

**quit.addActionListener(new ActionListener() {**

**public void actionPerformed(ActionEvent e) { System.exit(0); }**

**});**

**// In addition to the Quit menu item, also handle window close events**

**this.addWindowListener(new WindowAdapter() {**

**public void windowClosing(WindowEvent e) { System.exit(0); }**

**});**

**// Insert each of the example objects into the tabbed pane**

**for(int i = 0; i < examples.length; i++) {**

**GraphSample e = examples[i];**

**tpane.addTab(e.getName(), new GraphSamplePane(e));**

**}**

**}**

**/\*\***

**\* This inner class is a custom Swing component that displays**

**\* a GraphSample object.**

**\*/**

**public class GraphSamplePane extends JComponent {**

**GraphSample example; // The example to display**

**Dimension size; // How much space it requires**

**public GraphSamplePane(GraphSample example) {**

**this.example = example;**

**size = new Dimension(example.getWidth(), example.getHeight());**

**setMaximumSize( size );**

**}**

**/\*\* Draw the component and the example it contains \*/**

**public void paintComponent(Graphics g) {**

**g.setColor(Color.white); // set the background**

**g.fillRect(0, 0, size.width, size.height); // to white**

**g.setColor(Color.black); // set a default drawing color**

**example.draw((Graphics2D) g, this); // ask example to draw itself**

**}**

**// These methods specify how big the component must be**

**public Dimension getPreferredSize() { return size; }**

**public Dimension getMinimumSize() { return size; }**

**}**

**public static void main(String[] args) {**

**GraphSample[] examples = new GraphSample[1];**

**// Try to instantiate the named GraphSample class**

**try {**

**Class exampleClass = Class.forName(classname);**

**examples[0] = (GraphSample) exampleClass.newInstance();**

**}**

**catch (ClassNotFoundException e) { // unknown class**

**System.err.println("Couldn't find example: " + classname);**

**System.exit(1);**

**}**

**catch (ClassCastException e) { // wrong type of class**

**System.err.println("Class " + classname +**

**" is not a GraphSample");**

**System.exit(1);**

**}**

**catch (Exception e) { // class doesn't have a public constructor**

**// catch InstantiationException, IllegalAccessException**

**System.err.println("Couldn't instantiate example: " +**

**classname);**

**System.exit(1);**

**}**

**// Now create a window to display the examples in, and make it visible**

**GraphSampleFrame f = new GraphSampleFrame(examples);**

**f.pack();**

**f.setVisible(true);**

**}**

**}**

**////////////////////////////////////////////////////////////////////////////**

**// interface GraphSample**

**////////////////////////////////////////////////////////////////////////////**

**/\*\***

**\* This interface defines the methods that must be implemented by an**

**\* object that is to be displayed by the GraphSampleFrame object**

**\*/**

**interface GraphSample {**

**public String getName(); // Return the example name**

**public int getWidth(); // Return its width**

**public int getHeight(); // Return its height**

**public void draw(Graphics2D g, Component c); // Draw the example**

**}**

**A.3 Исходные коды лабораторной работы №3**

**Main.java**

**import java.awt.\*;**

**import java.awt.event.\*;**

**import javax.swing.\*;**

**import javax.swing.event.\*;**

**import java.awt.print.\*;**

**import java.awt.geom.\*;**

**import java.awt.font.\*;**

**/\*\* A demonstration of writing custom Stroke classes \*/**

**public class Main implements GraphSample {**

**static final int WIDTH = 500, HEIGHT = 500; // Size of our example**

**public String getName() {return "Custom Strokes";} // From GraphSample**

**public int getWidth() { return WIDTH; } // From GraphSample**

**public int getHeight() { return HEIGHT; } // From GraphSample**

**// These are the various stroke objects we'll demonstrate**

**Stroke stroke = new MyStroke(2.0f, 8.0f, 8.0f);**

**Shape shape = new MyShape(0,0,400,400);**

**/\*\* Draw the example \*/**

**public void draw(Graphics2D g, Component c) {**

**g.setColor(Color.black);**

**g.setRenderingHint(RenderingHints.KEY\_ANTIALIASING,**

**RenderingHints.VALUE\_ANTIALIAS\_ON);**

**g.translate(20, 20);**

**g.setStroke(stroke);**

**g.draw(shape);**

**}**

**}**

**class MyStroke implements Stroke {**

**private float amplitude = 10.0f;**

**private float wavelength = 10.0f;**

**private Stroke stroke;**

**private static final float FLATNESS = 1;**

**public MyStroke(float stroke,float amplitude, float wavelength ) {**

**this.stroke = new BasicStroke(stroke);**

**this.amplitude = amplitude;**

**this.wavelength = wavelength;**

**}**

**public Shape createStrokedShape( Shape shape ) {**

**GeneralPath result = new GeneralPath();**

**PathIterator it = new FlatteningPathIterator( shape.getPathIterator( null ), FLATNESS );**

**float points[] = new float[6];**

**float moveX = 0, moveY = 0;**

**float lastX = 0, lastY = 0;**

**float thisX = 0, thisY = 0;**

**int type = 0;**

**boolean first = false;**

**float next = 0;**

**int phase = 0;**

**float factor = 1;**

**while ( !it.isDone() ) {**

**type = it.currentSegment( points );**

**switch( type ){**

**case PathIterator.SEG\_MOVETO:**

**moveX = lastX = points[0];**

**moveY = lastY = points[1];**

**result.moveTo( moveX, moveY );**

**first = true;**

**next = wavelength/2;**

**break;**

**case PathIterator.SEG\_CLOSE:**

**points[0] = moveX;**

**points[1] = moveY;**

**// Fall into....**

**case PathIterator.SEG\_LINETO:**

**thisX = points[0];**

**thisY = points[1];**

**float dx = thisX-lastX;**

**float dy = thisY-lastY;**

**float distance = (float)Math.sqrt( dx\*dx + dy\*dy );**

**if ( distance >= next ) {**

**float r = 1.0f/distance;**

**float angle = (float)Math.atan2( dy, dx );**

**while ( distance >= next ) {**

**float x = lastX + next\*dx\*r;**

**float y = lastY + next\*dy\*r;**

**float tx = amplitude\*dy\*r;**

**float ty = amplitude\*dx\*r;**

**double ux=tx\*Math.abs(Math.sin(angle));**

**double uy=ty\*Math.abs(Math.cos(angle));**

**double z=Math.signum(dy/dx);**

**switch(phase)**

**{**

**case 1:case 4:**

**result.lineTo( x, y );**

**break;**

**case 2:case 5:**

**result.quadTo(x+z\*ux/2-2\*amplitude/4, y-2\*z\*uy/2+z\*amplitude/4, x+z\*ux, y-z\*uy);**

**break;**

**case 3:case 0:**

**result.quadTo(x+z\*ux/2+z\*amplitude/4, y-z\*uy/2-z\*amplitude/4, x, y);**

**break;**

**}**

**next += wavelength;**

**phase++;**

**phase=phase%6;**

**}**

**}**

**next -= distance;**

**first = false;**

**lastX = thisX;**

**lastY = thisY;**

**if ( type == PathIterator.SEG\_CLOSE )**

**result.closePath();**

**break;**

**}**

**it.next();**

**}**

**return stroke.createStrokedShape( result );**

**}**

**}**

**class MyShape implements Shape {**

**double x;**

**double y;**

**double width;**

**double height;**

**/\*\***

**\* The constructor. It takes arguments for the center of the shape, the**

**\* start point, and the end point. The start and end points are specified**

**\* in terms of angle and radius. The spiral curve is formed by varying**

**\* the angle and radius smoothly between the two end points.**

**\*\*/**

**public MyShape(double x,double y,double width,double height)**

**{**

**this.x=x;**

**this.y=y;**

**this.width=width;**

**this.height=height;**

**}**

**/\*\***

**\* The bounding box of a Spiral is the same as the bounding box of a**

**\* circle with the same center and the maximum radius**

**\*\*/**

**public Rectangle getBounds() {**

**return new Rectangle((int)(x),**

**(int)(y),**

**(int)(width), (int)(height));**

**}**

**/\*\* Same as getBounds(), but with floating-point coordinates \*/**

**public Rectangle2D getBounds2D() {**

**return new Rectangle2D.Double(x, y,**

**width, height);**

**}**

**/\*\***

**\* A spiral is an open curve, not a not a closed area; it does not have an**

**\* inside and an outsize, so the contains() methods always return false.**

**\*\*/**

**public boolean contains(double x, double y) { return false; }**

**public boolean contains(Point2D p) { return false; }**

**public boolean contains(Rectangle2D r) { return false; }**

**public boolean contains(double x, double y, double w, double h) {**

**return false;**

**}**

**/\*\***

**\* This method is allowed to approximate if it would be too computationally**

**\* intensive to determine an exact answer. Therefore, we check whether**

**\* the rectangle intersects a circle of the outer radius. This is a good**

**\* guess for a tight spiral, but less good for a "loose" spiral.**

**\*\*/**

**public boolean intersects(double x, double y, double w, double h) {**

**Shape approx = new Rectangle2D.Double(x, y,**

**width, height);**

**return approx.intersects(x, y, w, h);**

**}**

**/\*\* This version of intersects() just calls the one above \*/**

**public boolean intersects(Rectangle2D r) {**

**return intersects(r.getX(), r.getY(), r.getWidth(), r.getHeight());**

**}**

**/\*\***

**\* This method is the heart of all Shape implementations. It returns a**

**\* PathIterator that describes the shape in terms of the line and curve**

**\* segments that comprise it. Our iterator implementation approximates**

**\* the shape of the spiral using line segments only. We pass in a**

**\* "flatness" argument that tells it how good the approximation must be.**

**\* (smaller numbers mean a better approximation).**

**\*/**

**public PathIterator getPathIterator(AffineTransform at) {**

**return new SpiralIterator(at, width/500.0);**

**}**

**/\*\***

**\* Return a PathIterator that describes the shape in terms of line**

**\* segments only, with an approximation quality specified by flatness.**

**\*\*/**

**public PathIterator getPathIterator(AffineTransform at, double flatness) {**

**return new SpiralIterator(at, flatness);**

**}**

**/\*\***

**\* This inner class is the PathIterator for our Spiral shape. For**

**\* simplicity, it does not describe the spiral path in terms of Bezier**

**\* curve segments, but simply approximates it with line segments. The**

**\* flatness property specifies how far the approximation is allowed to**

**\* deviate from the true curve.**

**\*\*/**

**class SpiralIterator implements PathIterator {**

**AffineTransform transform;**

**double flatness;**

**double angle\_start = -Math.PI/2+0.0001;**

**double angle = angle\_start;**

**double a = 6000;**

**boolean done = false;**

**/\*\* A simple constructor. Just store the parameters into fields \*/**

**public SpiralIterator(AffineTransform transform, double flatness) {**

**this.transform = transform;**

**this.flatness = flatness;**

**}**

**/\*\***

**\* All PathIterators have a "winding rule" that helps to specify what**

**\* is the inside of a area and what is the outside. If you fill a**

**\* spiral (which you're not supposed to do) the winding rule returned**

**\* here yields better results than the alternative, WIND\_EVEN\_ODD**

**\*\*/**

**public int getWindingRule() { return WIND\_NON\_ZERO; }**

**/\*\* Returns true if the entire path has been iterated \*/**

**public boolean isDone() { return done; }**

**/\*\***

**\* Store the coordinates of the current segment of the path into the**

**\* specified array, and return the type of the segment. Use**

**\* trigonometry to compute the coordinates based on the current angle**

**\* and radius. If this was the first point, return a MOVETO segment,**

**\* otherwise return a LINETO segment. Also, check to see if we're done.**

**\*\*/**

**public int currentSegment(float[] coords) {**

**if(done)**

**return SEG\_CLOSE;**

**double r=Math.sqrt(Math.abs(a/Math.tan(angle)));**

**coords[0] = (float)(x+width/2+r\*Math.cos(angle));**

**coords[1] = (float)(x+height/2+r\*Math.sin(angle));**

**if (transform != null) transform.transform(coords, 0, coords, 0,1);**

**if (angle >= Math.PI\*3/2-0.0001){done = true;}**

**if (angle == angle\_start) return SEG\_MOVETO;**

**return SEG\_LINETO;**

**}**

**/\*\* This method is the same as above, except using double values \*/**

**public int currentSegment(double[] coords) {**

**if(done)**

**return SEG\_CLOSE;**

**double r=Math.sqrt(Math.abs(a/Math.tan(angle)));**

**coords[0] = (double)(x+width/2+r\*Math.cos(angle));**

**coords[1] = (double)(x+height/2+r\*Math.sin(angle));**

**if (transform != null) transform.transform(coords, 0, coords, 0,1);**

**if (angle >= Math.PI\*3/2-0.0001){done = true;}**

**if (angle == angle\_start) return SEG\_MOVETO;**

**return SEG\_LINETO;**

**}**

**/\*\***

**\* Move on to the next segment of the path. Compute the angle and**

**\* radius values for the next point in the spiral.**

**\*\*/**

**public void next() {**

**if (done) return;**

**angle+=0.01;**

**if (angle >= Math.PI\*3/2-0.0001){done = true;}**

**}**

**}**

**}**

**////////////////////////////////////////////////////////////////////////////**

**// Frame**

**////////////////////////////////////////////////////////////////////////////**

**class GraphSampleFrame extends JFrame {**

**// The class name of the requested example**

**static final String classname = "Main";**

**public GraphSampleFrame(final GraphSample[] examples) {**

**super("GraphSampleFrame");**

**Container cpane = getContentPane(); // Set up the frame**

**cpane.setLayout(new BorderLayout());**

**final JTabbedPane tpane = new JTabbedPane(); // And the tabbed pane**

**cpane.add(tpane, BorderLayout.CENTER);**

**// Add a menubar**

**JMenuBar menubar = new JMenuBar(); // Create the menubar**

**this.setJMenuBar(menubar); // Add it to the frame**

**JMenu filemenu = new JMenu("File"); // Create a File menu**

**menubar.add(filemenu); // Add to the menubar**

**JMenuItem quit = new JMenuItem("Quit"); // Create a Quit item**

**filemenu.add(quit); // Add it to the menu**

**// Tell the Quit menu item what to do when selected**

**quit.addActionListener(new ActionListener() {**

**public void actionPerformed(ActionEvent e) { System.exit(0); }**

**});**

**// In addition to the Quit menu item, also handle window close events**

**this.addWindowListener(new WindowAdapter() {**

**public void windowClosing(WindowEvent e) { System.exit(0); }**

**});**

**// Insert each of the example objects into the tabbed pane**

**for(int i = 0; i < examples.length; i++) {**

**GraphSample e = examples[i];**

**tpane.addTab(e.getName(), new GraphSamplePane(e));**

**}**

**}**

**/\*\***

**\* This inner class is a custom Swing component that displays**

**\* a GraphSample object.**

**\*/**

**public class GraphSamplePane extends JComponent {**

**GraphSample example; // The example to display**

**Dimension size; // How much space it requires**

**public GraphSamplePane(GraphSample example) {**

**this.example = example;**

**size = new Dimension(example.getWidth(), example.getHeight());**

**setMaximumSize( size );**

**}**

**/\*\* Draw the component and the example it contains \*/**

**public void paintComponent(Graphics g) {**

**g.setColor(Color.white); // set the background**

**g.fillRect(0, 0, size.width, size.height); // to white**

**g.setColor(Color.black); // set a default drawing color**

**example.draw((Graphics2D) g, this); // ask example to draw itself**

**}**

**// These methods specify how big the component must be**

**public Dimension getPreferredSize() { return size; }**

**public Dimension getMinimumSize() { return size; }**

**}**

**public static void main(String[] args) {**

**GraphSample[] examples = new GraphSample[1];**

**// Try to instantiate the named GraphSample class**

**try {**

**Class exampleClass = Class.forName(classname);**

**examples[0] = (GraphSample) exampleClass.newInstance();**

**}**

**catch (ClassNotFoundException e) { // unknown class**

**System.err.println("Couldn't find example: " + classname);**

**System.exit(1);**

**}**

**catch (ClassCastException e) { // wrong type of class**

**System.err.println("Class " + classname +**

**" is not a GraphSample");**

**System.exit(1);**

**}**

**catch (Exception e) { // class doesn't have a public constructor**

**// catch InstantiationException, IllegalAccessException**

**System.err.println("Couldn't instantiate example: " +**

**classname);**

**System.exit(1);**

**}**

**// Now create a window to display the examples in, and make it visible**

**GraphSampleFrame f = new GraphSampleFrame(examples);**

**f.pack();**

**f.setVisible(true);**

**}**

**}**

**////////////////////////////////////////////////////////////////////////////**

**// interface GraphSample**

**////////////////////////////////////////////////////////////////////////////**

**/\*\***

**\* This interface defines the methods that must be implemented by an**

**\* object that is to be displayed by the GraphSampleFrame object**

**\*/**

**interface GraphSample {**

**public String getName(); // Return the example name**

**public int getWidth(); // Return its width**

**public int getHeight(); // Return its height**

**public void draw(Graphics2D g, Component c); // Draw the example**

**}**

**A.4 Исходные коды лабораторной работы №4**

**Main.java**

**import java.awt.\*;**

**import java.awt.JobAttributes.SidesType;**

**import java.awt.event.\*;**

**import javax.swing.\*;**

**import javax.swing.event.\*;**

**import java.awt.print.\*;**

**import java.awt.geom.\*;**

**import java.awt.image.BufferedImage;**

**import java.awt.image.ImageObserver;**

**import java.awt.font.\*;**

**import java.io.BufferedReader;**

**import java.io.File;**

**import java.io.FileInputStream;**

**import java.io.FileNotFoundException;**

**import java.io.FileReader;**

**import java.io.IOException;**

**import java.io.PrintWriter;**

**import java.io.Writer;**

**import java.nio.file.Files;**

**import java.nio.file.Path;**

**import java.nio.file.Paths;**

**import java.text.DateFormat;**

**import java.util.Date;**

**import java.util.Properties;**

**import java.util.Scanner;**

**import java.util.TimeZone;**

**/\*\* A demonstration of writing custom Stroke classes \*/**

**public class Main implements GraphSample {**

**static final int WIDTH = 500, HEIGHT = 500; // Size of our example**

**public String getName() {**

**return "Custom Strokes";**

**} // From GraphSample**

**public int getWidth() {**

**return WIDTH;**

**} // From GraphSample**

**public int getHeight() {**

**return HEIGHT;**

**} // From GraphSample**

**// These are the various stroke objects we'll demonstrate**

**Stroke stroke = new MyStroke(2.0f, 8.0f, 8.0f);**

**Shape shape = new MyShape(0, 0, 400, 400);**

**/\*\* Draw the example \*/**

**public void draw(Graphics2D g, Component c) {**

**g.setColor(Color.black);**

**g.setRenderingHint(RenderingHints.KEY\_ANTIALIASING,**

**RenderingHints.VALUE\_ANTIALIAS\_ON);**

**g.translate(20, 20);**

**g.setStroke(stroke);**

**g.draw(shape);**

**}**

**}**

**class MyStroke implements Stroke {**

**private float amplitude = 10.0f;**

**private float wavelength = 10.0f;**

**private Stroke stroke;**

**private static final float FLATNESS = 1;**

**public MyStroke(float stroke, float amplitude, float wavelength) {**

**this.stroke = new BasicStroke(stroke);**

**this.amplitude = amplitude;**

**this.wavelength = wavelength;**

**}**

**public Shape createStrokedShape(Shape shape) {**

**GeneralPath result = new GeneralPath();**

**PathIterator it = new FlatteningPathIterator(**

**shape.getPathIterator(null), FLATNESS);**

**float points[] = new float[6];**

**float moveX = 0, moveY = 0;**

**float lastX = 0, lastY = 0;**

**float thisX = 0, thisY = 0;**

**int type = 0;**

**boolean first = false;**

**float next = 0;**

**int phase = 0;**

**float factor = 1;**

**while (!it.isDone()) {**

**type = it.currentSegment(points);**

**switch (type) {**

**case PathIterator.SEG\_MOVETO:**

**moveX = lastX = points[0];**

**moveY = lastY = points[1];**

**result.moveTo(moveX, moveY);**

**first = true;**

**next = wavelength / 2;**

**break;**

**case PathIterator.SEG\_CLOSE:**

**points[0] = moveX;**

**points[1] = moveY;**

**// Fall into....**

**case PathIterator.SEG\_LINETO:**

**thisX = points[0];**

**thisY = points[1];**

**float dx = thisX - lastX;**

**float dy = thisY - lastY;**

**float distance = (float) Math.sqrt(dx \* dx + dy \* dy);**

**if (distance >= next) {**

**float r = 1.0f / distance;**

**float angle = (float) Math.atan2(dy, dx);**

**while (distance >= next) {**

**float x = lastX + next \* dx \* r;**

**float y = lastY + next \* dy \* r;**

**float tx = amplitude \* dy \* r;**

**float ty = amplitude \* dx \* r;**

**double ux=tx\*Math.abs(Math.sin(angle));**

**double uy=ty\*Math.abs(Math.cos(angle));**

**double z=Math.signum(dy/dx);**

**switch (phase) {**

**case 1:case 4:**

**result.lineTo( x, y );**

**break;**

**case 2:case 5:**

**result.quadTo(x+z\*ux/2-2\*amplitude/4, y-2\*z\*uy/2+z\*amplitude/4, x+z\*ux, y-z\*uy);**

**break;**

**case 3:case 0:**

**result.quadTo(x+z\*ux/2+z\*amplitude/4, y-z\*uy/2-z\*amplitude/4, x, y);**

**break;**

**}**

**next += wavelength;**

**phase++;**

**phase = phase % 6;**

**}**

**}**

**next -= distance;**

**first = false;**

**lastX = thisX;**

**lastY = thisY;**

**if (type == PathIterator.SEG\_CLOSE)**

**result.closePath();**

**break;**

**}**

**it.next();**

**}**

**return stroke.createStrokedShape(result);**

**}**

**}**

**class MyShape implements Shape {**

**double x;**

**double y;**

**double width;**

**double height;**

**/\*\***

**\* The constructor. It takes arguments for the center of the shape, the**

**\* start point, and the end point. The start and end points are specified**

**\* in terms of angle and radius. The spiral curve is formed by varying**

**\* the angle and radius smoothly between the two end points.**

**\*\*/**

**public MyShape(double x,double y,double width,double height)**

**{**

**this.x=x;**

**this.y=y;**

**this.width=width;**

**this.height=height;**

**}**

**/\*\***

**\* The bounding box of a Spiral is the same as the bounding box of a**

**\* circle with the same center and the maximum radius**

**\*\*/**

**public Rectangle getBounds() {**

**return new Rectangle((int)(x),**

**(int)(y),**

**(int)(width), (int)(height));**

**}**

**/\*\* Same as getBounds(), but with floating-point coordinates \*/**

**public Rectangle2D getBounds2D() {**

**return new Rectangle2D.Double(x, y,**

**width, height);**

**}**

**/\*\***

**\* A spiral is an open curve, not a not a closed area; it does not have an**

**\* inside and an outsize, so the contains() methods always return false.**

**\*\*/**

**public boolean contains(double x, double y) { return false; }**

**public boolean contains(Point2D p) { return false; }**

**public boolean contains(Rectangle2D r) { return false; }**

**public boolean contains(double x, double y, double w, double h) {**

**return false;**

**}**

**/\*\***

**\* This method is allowed to approximate if it would be too computationally**

**\* intensive to determine an exact answer. Therefore, we check whether**

**\* the rectangle intersects a circle of the outer radius. This is a good**

**\* guess for a tight spiral, but less good for a "loose" spiral.**

**\*\*/**

**public boolean intersects(double x, double y, double w, double h) {**

**Shape approx = new Rectangle2D.Double(x, y,**

**width, height);**

**return approx.intersects(x, y, w, h);**

**}**

**/\*\* This version of intersects() just calls the one above \*/**

**public boolean intersects(Rectangle2D r) {**

**return intersects(r.getX(), r.getY(), r.getWidth(), r.getHeight());**

**}**

**/\*\***

**\* This method is the heart of all Shape implementations. It returns a**

**\* PathIterator that describes the shape in terms of the line and curve**

**\* segments that comprise it. Our iterator implementation approximates**

**\* the shape of the spiral using line segments only. We pass in a**

**\* "flatness" argument that tells it how good the approximation must be.**

**\* (smaller numbers mean a better approximation).**

**\*/**

**public PathIterator getPathIterator(AffineTransform at) {**

**return new SpiralIterator(at, width/500.0);**

**}**

**/\*\***

**\* Return a PathIterator that describes the shape in terms of line**

**\* segments only, with an approximation quality specified by flatness.**

**\*\*/**

**public PathIterator getPathIterator(AffineTransform at, double flatness) {**

**return new SpiralIterator(at, flatness);**

**}**

**/\*\***

**\* This inner class is the PathIterator for our Spiral shape. For**

**\* simplicity, it does not describe the spiral path in terms of Bezier**

**\* curve segments, but simply approximates it with line segments. The**

**\* flatness property specifies how far the approximation is allowed to**

**\* deviate from the true curve.**

**\*\*/**

**class SpiralIterator implements PathIterator {**

**AffineTransform transform;**

**double flatness;**

**double angle\_start = -Math.PI/2+0.0001;**

**double angle = angle\_start;**

**double a = 6000;**

**boolean done = false;**

**/\*\* A simple constructor. Just store the parameters into fields \*/**

**public SpiralIterator(AffineTransform transform, double flatness) {**

**this.transform = transform;**

**this.flatness = flatness;**

**}**

**/\*\***

**\* All PathIterators have a "winding rule" that helps to specify what**

**\* is the inside of a area and what is the outside. If you fill a**

**\* spiral (which you're not supposed to do) the winding rule returned**

**\* here yields better results than the alternative, WIND\_EVEN\_ODD**

**\*\*/**

**public int getWindingRule() { return WIND\_NON\_ZERO; }**

**/\*\* Returns true if the entire path has been iterated \*/**

**public boolean isDone() { return done; }**

**/\*\***

**\* Store the coordinates of the current segment of the path into the**

**\* specified array, and return the type of the segment. Use**

**\* trigonometry to compute the coordinates based on the current angle**

**\* and radius. If this was the first point, return a MOVETO segment,**

**\* otherwise return a LINETO segment. Also, check to see if we're done.**

**\*\*/**

**public int currentSegment(float[] coords) {**

**if(done)**

**return SEG\_CLOSE;**

**double r=Math.sqrt(Math.abs(a/Math.tan(angle)));**

**coords[0] = (float)(x+width/2+r\*Math.cos(angle));**

**coords[1] = (float)(x+height/2+r\*Math.sin(angle));**

**if (transform != null) transform.transform(coords, 0, coords, 0,1);**

**if (angle >= Math.PI\*3/2-0.0001){done = true;}**

**if (angle == angle\_start) return SEG\_MOVETO;**

**return SEG\_LINETO;**

**}**

**/\*\* This method is the same as above, except using double values \*/**

**public int currentSegment(double[] coords) {**

**if(done)**

**return SEG\_CLOSE;**

**double r=Math.sqrt(Math.abs(a/Math.tan(angle)));**

**coords[0] = (double)(x+width/2+r\*Math.cos(angle));**

**coords[1] = (double)(x+height/2+r\*Math.sin(angle));**

**if (transform != null) transform.transform(coords, 0, coords, 0,1);**

**if (angle >= Math.PI\*3/2-0.0001){done = true;}**

**if (angle == angle\_start) return SEG\_MOVETO;**

**return SEG\_LINETO;**

**}**

**/\*\***

**\* Move on to the next segment of the path. Compute the angle and**

**\* radius values for the next point in the spiral.**

**\*\*/**

**public void next() {**

**if (done) return;**

**angle+=0.01;**

**if (angle >= Math.PI\*3/2-0.0001){done = true;}**

**}**

**}**

**}**

**class HardcopyWriter extends Writer {**

**// These are the instance variables for the class**

**protected PrintJob job; // The PrintJob object in use**

**protected Graphics page; // Graphics object for current page**

**protected String jobname; // The name of the print job**

**protected int fontsize; // Point size of the font**

**protected String time; // Current time (appears in header)**

**protected Dimension pagesize; // Size of the page (in dots)**

**protected int pagedpi; // Page resolution in dots per inch**

**protected Font font, headerfont; // Body font and header font**

**protected FontMetrics metrics; // Metrics for the body font**

**protected FontMetrics headermetrics; // Metrics for the header font**

**protected int x0, y0; // Upper-left corner inside margin**

**protected int width, height; // Size (in dots) inside margins**

**protected int headery; // Baseline of the page header**

**protected int charwidth; // The width of each character**

**protected int lineheight; // The height of each line**

**protected int lineascent; // Offset of font baseline**

**protected int chars\_per\_line; // Number of characters per line**

**protected int lines\_per\_page; // Number of lines per page**

**protected int charnum = 0, linenum = 0; // Current column and line position**

**protected int pagenum = 0; // Current page number**

**// A field to save state between invocations of the write() method**

**private boolean last\_char\_was\_return = false;**

**// A static variable that holds user preferences between print jobs**

**protected static JobAttributes jobattrs = new JobAttributes();**

**protected static PageAttributes pageattrs = new PageAttributes();**

**/\*\***

**\* The constructor for this class has a bunch of arguments: The frame**

**\* argument is required for all printing in Java. The jobname appears left**

**\* justified at the top of each printed page. The font size is specified in**

**\* points, as on-screen font sizes are. The margins are specified in inches**

**\* (or fractions of inches).**

**\*\*/**

**public HardcopyWriter(Frame frame, String jobname, int fontsize,**

**double leftmargin, double rightmargin, double topmargin,**

**double bottommargin) throws HardcopyWriter.PrintCanceledException {**

**// Get the PrintJob object with which we'll do all the printing.**

**// The call is synchronized on the static printprops object, which**

**// means that only one print dialog can be popped up at a time.**

**// If the user clicks Cancel in the print dialog, throw an exception.**

**Toolkit toolkit = frame.getToolkit(); // get Toolkit from Frame**

**synchronized (jobattrs) {**

**pageattrs.setOrientationRequested(PageAttributes.OrientationRequestedType.LANDSCAPE);**

**jobattrs.setSides(JobAttributes.SidesType.TWO\_SIDED\_LONG\_EDGE);**

**job = toolkit.getPrintJob(frame, jobname, jobattrs, pageattrs);**

**}**

**if (job == null)**

**throw new PrintCanceledException("User cancelled print request");**

**pagesize = job.getPageDimension(); // query the page size**

**pagedpi = job.getPageResolution(); // query the page resolution**

**// Bug Workaround:**

**// On windows, getPageDimension() and getPageResolution don't work, so**

**// we've got to fake them.**

**if (System.getProperty("os.name").regionMatches(true, 0, "windows", 0,**

**7)) {**

**// Use screen dpi, which is what the PrintJob tries to emulate**

**pagedpi = toolkit.getScreenResolution();**

**// Assume a 8.5" x 11" page size. A4 paper users must change this.**

**pagesize = new Dimension((int) (8.5 \* pagedpi), 11 \* pagedpi);**

**// We also have to adjust the fontsize. It is specified in points,**

**// (1 point = 1/72 of an inch) but Windows measures it in pixels.**

**fontsize = fontsize \* pagedpi / 72;**

**}**

**// Compute coordinates of the upper-left corner of the page.**

**// I.e. the coordinates of (leftmargin, topmargin). Also compute**

**// the width and height inside of the margins.**

**x0 = (int) (leftmargin \* pagedpi);**

**y0 = (int) (topmargin \* pagedpi);**

**width = pagesize.width - (int) ((leftmargin + rightmargin) \* pagedpi);**

**height = pagesize.height - (int) ((topmargin + bottommargin) \* pagedpi);**

**// Get body font and font size**

**font = new Font("Monospaced", Font.PLAIN, fontsize);**

**metrics = frame.getFontMetrics(font);**

**lineheight = metrics.getHeight();**

**lineascent = metrics.getAscent();**

**charwidth = metrics.charWidth('0'); // Assumes a monospaced font!**

**// Now compute columns and lines will fit inside the margins**

**chars\_per\_line = width / charwidth;**

**lines\_per\_page = height / lineheight;**

**// Get header font information**

**// And compute baseline of page header: 1/8" above the top margin**

**headerfont = new Font("SansSerif", Font.ITALIC, fontsize);**

**headermetrics = frame.getFontMetrics(headerfont);**

**headery = y0 - (int) (0.125 \* pagedpi) - headermetrics.getHeight()**

**+ headermetrics.getAscent();**

**// Compute the date/time string to display in the page header**

**DateFormat df = DateFormat.getDateTimeInstance(DateFormat.LONG,**

**DateFormat.SHORT);**

**df.setTimeZone(TimeZone.getDefault());**

**time = df.format(new Date());**

**this.jobname = jobname; // save name**

**this.fontsize = fontsize; // save font size**

**}**

**/\*\***

**\* This is the write() method of the stream. All Writer subclasses implement**

**\* this. All other versions of write() are variants of this one**

**\*\*/**

**public void write(char[] buffer, int index, int len) {**

**synchronized (this.lock) { // For thread safety**

**// Loop through all the characters passed to us**

**for (int i = index; i < index + len; i++) {**

**// If we haven't begun a page (or a new page), do that now.**

**if (page == null)**

**newpage();**

**// If the character is a line terminator, then begin new line,**

**// unless it is a \n immediately after a \r.**

**if (buffer[i] == '\n') {**

**if (!last\_char\_was\_return)**

**newline();**

**continue;**

**}**

**if (buffer[i] == '\r') {**

**newline();**

**last\_char\_was\_return = true;**

**continue;**

**} else**

**last\_char\_was\_return = false;**

**// If it some other non-printing character, ignore it.**

**if (Character.isWhitespace(buffer[i])**

**&& !Character.isSpaceChar(buffer[i])**

**&& (buffer[i] != '\t'))**

**continue;**

**// If no more characters will fit on the line, start new line.**

**if (charnum >= chars\_per\_line) {**

**newline();**

**// Also start a new page, if necessary**

**if (page == null)**

**newpage();**

**}**

**// Now print the character:**

**// If it is a space, skip one space, without output.**

**// If it is a tab, skip the necessary number of spaces.**

**// Otherwise, print the character.**

**// It is inefficient to draw only one character at a time, but**

**// because our FontMetrics don't match up exactly to what the**

**// printer uses we need to position each character individually**

**if (Character.isSpaceChar(buffer[i]))**

**charnum++;**

**else if (buffer[i] == '\t')**

**charnum += 8 - (charnum % 8);**

**else {**

**page.drawChars(buffer, i, 1, x0 + charnum \* charwidth, y0**

**+ (linenum \* lineheight) + lineascent);**

**charnum++;**

**}**

**}**

**}**

**}**

**public void printFigure(ImageObserver obs)**

**{**

**Stroke stroke = new MyStroke(2.0f, 8.0f, 8.0f);**

**Shape shape = new MyShape(0, 0, 280, 280);**

**BufferedImage image = new BufferedImage(400, 400, BufferedImage.TYPE\_INT\_RGB);**

**Graphics2D g = image.createGraphics();**

**g.setColor(Color.white);**

**g.setBackground(Color.white);**

**g.fillRect(0,0,400,400);**

**g.setColor(Color.black);**

**g.setRenderingHint(RenderingHints.KEY\_ANTIALIASING,**

**RenderingHints.VALUE\_ANTIALIAS\_ON);**

**g.translate(50, 50);**

**g.setStroke(stroke);**

**g.draw(shape);**

**Image img=image.getScaledInstance(200, 200, Image.SCALE\_SMOOTH);**

**page.drawImage(img, 300, 120, 200, 200,obs);**

**page.translate(0, 220);**

**}**

**/\*\***

**\* This is the flush() method that all Writer subclasses must implement.**

**\* There is no way to flush a PrintJob without prematurely printing the**

**\* page, so we don't do anything.**

**\*\*/**

**public void flush() { /\* do nothing \*/**

**}**

**/\*\***

**\* This is the close() method that all Writer subclasses must implement.**

**\* Print the pending page (if any) and terminate the PrintJob.**

**\*/**

**public void close() {**

**synchronized (this.lock) {**

**if (page != null)**

**page.dispose(); // Send page to the printer**

**job.end(); // Terminate the job**

**}**

**}**

**/\*\***

**\* Set the font style. The argument should be one of the font style**

**\* constants defined by the java.awt.Font class. All subsequent output will**

**\* be in that style. This method relies on all styles of the Monospaced font**

**\* having the same metrics.**

**\*\*/**

**public void setFontStyle(int style, int fontsize) {**

**synchronized (this.lock) {**

**// Try to set a new font, but restore current one if it fails**

**Font current = font;**

**try {**

**this.fontsize = fontsize;**

**font = new Font("Monospaced", style, fontsize);**

**} catch (Exception e) {**

**font = current;**

**}**

**// If a page is pending, set the new font. Otherwise newpage() will**

**if (page != null)**

**page.setFont(font);**

**}**

**}**

**/\*\* End the current page. Subsequent output will be on a new page. \*/**

**public void pageBreak() {**

**synchronized (this.lock) {**

**newpage();**

**}**

**}**

**/\*\* Return the number of columns of characters that fit on the page \*/**

**public int getCharactersPerLine() {**

**return this.chars\_per\_line;**

**}**

**/\*\* Return the number of lines that fit on a page \*/**

**public int getLinesPerPage() {**

**return this.lines\_per\_page;**

**}**

**/\*\* This internal method begins a new line \*/**

**protected void newline() {**

**charnum = 0; // Reset character number to 0**

**linenum++; // Increment line number**

**if (linenum >= lines\_per\_page) { // If we've reached the end of page**

**page.dispose(); // send page to printer**

**page = null; // but don't start a new page yet.**

**}**

**}**

**/\*\* This internal method begins a new page and prints the header. \*/**

**protected void newpage() {**

**page = job.getGraphics(); // Begin the new page**

**linenum = 0;**

**charnum = 0; // Reset line and char number**

**pagenum++; // Increment page number**

**page.setFont(headerfont); // Set the header font.**

**page.drawString(jobname, x0, headery); // Print job name left justified**

**String s = "- " + pagenum + " -"; // Print the page # centered.**

**int w = headermetrics.stringWidth(s);**

**page.drawString(s, x0 + (this.width - w) / 2, headery);**

**w = headermetrics.stringWidth(time); // Print date right justified**

**page.drawString(time, x0 + width - w, headery);**

**// Draw a line beneath the header**

**int y = headery + headermetrics.getDescent() + 1;**

**page.drawLine(x0, y, x0 + width, y);**

**// Set the basic monospaced font for the rest of the page.**

**page.setFont(font);**

**}**

**/\*\***

**\* This is the exception class that the HardcopyWriter constructor throws**

**\* when the user clicks "Cancel" in the print dialog box.**

**\*\*/**

**public static class PrintCanceledException extends Exception {**

**public PrintCanceledException(String msg) {**

**super(msg);**

**}**

**}**

**}**

**// //////////////////////////////////////////////////////////////////////////**

**// Frame**

**// //////////////////////////////////////////////////////////////////////////**

**class GraphSampleFrame extends JFrame {**

**// The class name of the requested example**

**static final String classname = "Main";**

**static GraphSampleFrame \_this = null;**

**public GraphSampleFrame(final GraphSample[] examples) {**

**super("GraphSampleFrame");**

**\_this=this;**

**Container cpane = getContentPane(); // Set up the frame**

**cpane.setLayout(new BorderLayout());**

**final JTabbedPane tpane = new JTabbedPane(); // And the tabbed pane**

**cpane.add(tpane, BorderLayout.CENTER);**

**// Add a menubar**

**JMenuBar menubar = new JMenuBar(); // Create the menubar**

**this.setJMenuBar(menubar); // Add it to the frame**

**JMenu filemenu = new JMenu("File"); // Create a File menu**

**menubar.add(filemenu); // Add to the menubar**

**JMenuItem print = new JMenuItem("Print"); // Create a Quit item**

**filemenu.add(print); // Add it to the menu**

**JMenuItem quit = new JMenuItem("Quit"); // Create a Quit item**

**filemenu.add(quit); // Add it to the menu**

**// Tell the Quit menu item what to do when selected**

**quit.addActionListener(new ActionListener() {**

**public void actionPerformed(ActionEvent e) {**

**System.exit(0);**

**}**

**});**

**print.addActionListener(new ActionListener() {**

**public void actionPerformed(ActionEvent e) {**

**HardcopyWriter hw;**

**try { hw=new HardcopyWriter(\_this, "Kappa",6,.75,.75,.75,.75);}**

**catch (HardcopyWriter.PrintCanceledException ev) { return; }**

**PrintWriter out = new PrintWriter(hw);**

**out.println("Kappa");**

**hw.printFigure(\_this);**

**String content = "";**

**try {**

**content = new String(Files.readAllBytes(Paths.get("in.txt")));**

**} catch (IOException e1) {**

**// TODO Auto-generated catch block**

**e1.printStackTrace();**

**}**

**out.println(content);**

**out.close();**

**hw.close();**

**}**

**});**

**// In addition to the Quit menu item, also handle window close events**

**this.addWindowListener(new WindowAdapter() {**

**public void windowClosing(WindowEvent e) {**

**System.exit(0);**

**}**

**});**

**// Insert each of the example objects into the tabbed pane**

**for (int i = 0; i < examples.length; i++) {**

**GraphSample e = examples[i];**

**tpane.addTab(e.getName(), new GraphSamplePane(e));**

**}**

**}**

**/\*\***

**\* This inner class is a custom Swing component that displays a GraphSample**

**\* object.**

**\*/**

**public class GraphSamplePane extends JComponent {**

**GraphSample example; // The example to display**

**Dimension size; // How much space it requires**

**public GraphSamplePane(GraphSample example) {**

**this.example = example;**

**size = new Dimension(example.getWidth(), example.getHeight());**

**setMaximumSize(size);**

**}**

**/\*\* Draw the component and the example it contains \*/**

**public void paintComponent(Graphics g) {**

**g.setColor(Color.white); // set the background**

**g.fillRect(0, 0, size.width, size.height); // to white**

**g.setColor(Color.black); // set a default drawing color**

**example.draw((Graphics2D) g, this); // ask example to draw itself**

**}**

**// These methods specify how big the component must be**

**public Dimension getPreferredSize() {**

**return size;**

**}**

**public Dimension getMinimumSize() {**

**return size;**

**}**

**}**

**public static void main(String[] args) {**

**GraphSample[] examples = new GraphSample[1];**

**// Try to instantiate the named GraphSample class**

**try {**

**Class exampleClass = Class.forName(classname);**

**examples[0] = (GraphSample) exampleClass.newInstance();**

**} catch (ClassNotFoundException e) { // unknown class**

**System.err.println("Couldn't find example: " + classname);**

**System.exit(1);**

**} catch (ClassCastException e) { // wrong type of class**

**System.err.println("Class " + classname + " is not a GraphSample");**

**System.exit(1);**

**} catch (Exception e) { // class doesn't have a public constructor**

**// catch InstantiationException, IllegalAccessException**

**System.err.println("Couldn't instantiate example: " + classname);**

**System.exit(1);**

**}**

**// Now create a window to display the examples in, and make it visible**

**GraphSampleFrame f = new GraphSampleFrame(examples);**

**f.pack();**

**f.setVisible(true);**

**}**

**}**

**// //////////////////////////////////////////////////////////////////////////**

**// interface GraphSample**

**// //////////////////////////////////////////////////////////////////////////**

**/\*\***

**\* This interface defines the methods that must be implemented by an object that**

**\* is to be displayed by the GraphSampleFrame object**

**\*/**

**interface GraphSample {**

**public String getName(); // Return the example name**

**public int getWidth(); // Return its width**

**public int getHeight(); // Return its height**

**public void draw(Graphics2D g, Component c); // Draw the example**

**}**

**A.5 Исходные коды лабораторной работы №5**

**Scribble.java**

**import java.awt.\*;**

**import java.awt.geom.\*;**

**import java.awt.datatransfer.\*;**

**import java.io.IOException;**

**import java.io.Serializable;**

**import java.util.StringTokenizer;**

**/\*\***

**\* This class represents a scribble composed of any number of "polylines".**

**\* Each "polyline" is set of connected line segments. A scribble is created**

**\* through a series of calls to the moveto() and lineto() methods. moveto()**

**\* specifies the starting point of a new polyline, and lineto() adds a new**

**\* point to the end of the current polyline().**

**\***

**\* This class implements the Shape interface which means that it can be drawn**

**\* using the Java2D graphics API**

**\***

**\* It also implements the Transferable interface, which means that it can**

**\* easily be used with cut-and-paste and drag-and-drop. It defines a custom**

**\* DataFlavor, scribbleDataFlavor, which transfers Scribble objects as Java**

**\* objects. However, it also supports cut-and-paste and drag-and-drop based**

**\* on a portable string representation of the scribble. The toString()**

**\* and parse() methods write and read this string format**

**\*\*/**

**class MyStroke implements Stroke {**

**private float amplitude = 10.0f;**

**private float wavelength = 10.0f;**

**private Stroke stroke;**

**private static final float FLATNESS = 1;**

**public MyStroke(float stroke,float amplitude, float wavelength ) {**

**this.stroke = new BasicStroke(stroke);**

**this.amplitude = amplitude;**

**this.wavelength = wavelength;**

**}**

**public Shape createStrokedShape( Shape shape ) {**

**GeneralPath result = new GeneralPath();**

**PathIterator it = new FlatteningPathIterator( shape.getPathIterator( null ), FLATNESS );**

**float points[] = new float[6];**

**float moveX = 0, moveY = 0;**

**float lastX = 0, lastY = 0;**

**float thisX = 0, thisY = 0;**

**int type = 0;**

**boolean first = false;**

**float next = 0;**

**int phase = 0;**

**float factor = 1;**

**while ( !it.isDone() ) {**

**type = it.currentSegment( points );**

**switch( type ){**

**case PathIterator.SEG\_MOVETO:**

**moveX = lastX = points[0];**

**moveY = lastY = points[1];**

**result.moveTo( moveX, moveY );**

**first = true;**

**next = wavelength/2;**

**break;**

**case PathIterator.SEG\_CLOSE:**

**points[0] = moveX;**

**points[1] = moveY;**

**// Fall into....**

**case PathIterator.SEG\_LINETO:**

**thisX = points[0];**

**thisY = points[1];**

**float dx = thisX-lastX;**

**float dy = thisY-lastY;**

**float distance = (float)Math.sqrt( dx\*dx + dy\*dy );**

**if ( distance >= next ) {**

**float r = 1.0f/distance;**

**float angle = (float)Math.atan2( dy, dx );**

**while ( distance >= next ) {**

**float x = lastX + next\*dx\*r;**

**float y = lastY + next\*dy\*r;**

**float tx = amplitude\*dy\*r;**

**float ty = amplitude\*dx\*r;**

**double ux=tx\*Math.abs(Math.sin(angle));**

**double uy=ty\*Math.abs(Math.cos(angle));**

**double z=Math.signum(dy/dx);**

**switch(phase)**

**{**

**case 1:case 4:**

**result.lineTo( x, y );**

**break;**

**case 2:case 5:**

**result.quadTo(x+z\*ux/2-2\*amplitude/4, y-2\*z\*uy/2+z\*amplitude/4, x+z\*ux, y-z\*uy);**

**break;**

**case 3:case 0:**

**result.quadTo(x+z\*ux/2+z\*amplitude/4, y-z\*uy/2-z\*amplitude/4, x, y);**

**break;**

**}**

**next += wavelength;**

**phase++;**

**phase=phase%6;**

**}**

**}**

**next -= distance;**

**first = false;**

**lastX = thisX;**

**lastY = thisY;**

**if ( type == PathIterator.SEG\_CLOSE )**

**result.closePath();**

**break;**

**}**

**it.next();**

**}**

**return stroke.createStrokedShape( result );**

**}**

**}**

**public class Scribble implements Shape, Transferable, Serializable, Cloneable {**

**protected double[] points = new double[64]; // The scribble data**

**protected int numPoints = 0; // The current number of points**

**double maxX = Double.NEGATIVE\_INFINITY; // The bounding box**

**double maxY = Double.NEGATIVE\_INFINITY;**

**double minX = Double.POSITIVE\_INFINITY;**

**double minY = Double.POSITIVE\_INFINITY;**

**/\*\***

**\* Begin a new polyline at (x,y). Note the use of Double.NaN in the**

**\* points array to mark the beginning of a new polyline**

**\*\*/**

**public void moveto(double x, double y) {**

**if (numPoints + 3 > points.length) reallocate();**

**// Mark this as the beginning of a new line**

**points[numPoints++] = Double.NaN;**

**// The rest of this method is just like lineto();**

**lineto(x, y);**

**}**

**/\*\***

**\* Add the point (x,y) to the end of the current polyline**

**\*\*/**

**public void lineto(double x, double y) {**

**if (numPoints + 2 > points.length) reallocate();**

**points[numPoints++] = x;**

**points[numPoints++] = y;**

**// See if the point enlarges our bounding box**

**if (x > maxX) maxX = x;**

**if (x < minX) minX = x;**

**if (y > maxY) maxY = y;**

**if (y < minY) minY = y;**

**}**

**public void draw(double x,double y,double width,double height)**

**{**

**double angle=-Math.PI/2+0.0001;**

**double a=6000;**

**double r,xpos,ypos;**

**r=Math.sqrt(Math.abs(a/Math.tan(angle)));**

**xpos = (double)(x+width/2+r\*Math.cos(angle));**

**ypos = (double)(y+height/2+r\*Math.sin(angle));**

**moveto(xpos,ypos);**

**while(angle<=Math.PI\*3/2-0.0001)**

**{**

**r=Math.sqrt(Math.abs(a/Math.tan(angle)));**

**xpos = (double)(x+width/2+r\*Math.cos(angle));**

**ypos = (double)(y+height/2+r\*Math.sin(angle));**

**lineto(xpos,ypos);**

**angle+=0.01;**

**}**

**}**

**/\*\***

**\* Append the Scribble s to this Scribble**

**\*\*/**

**public void append(Scribble s) {**

**int n = numPoints + s.numPoints;**

**double[] newpoints = new double[n];**

**System.arraycopy(points, 0, newpoints, 0, numPoints);**

**System.arraycopy(s.points, 0, newpoints, numPoints, s.numPoints);**

**points = newpoints;**

**numPoints = n;**

**minX = Math.min(minX, s.minX);**

**maxX = Math.max(maxX, s.maxX);**

**minY = Math.min(minY, s.minY);**

**maxY = Math.max(maxY, s.maxY);**

**}**

**/\*\***

**\* Translate the coordinates of all points in the Scribble by x,y**

**\*\*/**

**public void translate(double x, double y) {**

**for(int i = 0; i < numPoints; i++) {**

**if (Double.isNaN(points[i])) continue;**

**points[i++] += x;**

**points[i] += y;**

**}**

**minX += x; maxX += x;**

**minY += y; maxY += y;**

**}**

**/\*\* An internal method to make more room in the data array \*/**

**protected void reallocate() {**

**double[] newpoints = new double[points.length \* 2];**

**System.arraycopy(points, 0, newpoints, 0, numPoints);**

**points = newpoints;**

**}**

**/\*\* Clone a Scribble object and its internal array of data \*/**

**public Object clone() {**

**try {**

**Scribble s = (Scribble) super.clone(); // make a copy of all fields**

**s.points = (double[]) points.clone(); // copy the entire array**

**return s;**

**}**

**catch (CloneNotSupportedException e) { // This should never happen**

**return this;**

**}**

**}**

**/\*\* Convert the scribble data to a textual format \*/**

**public String toString() {**

**StringBuffer b = new StringBuffer();**

**for(int i = 0; i < numPoints; i++) {**

**if (Double.isNaN(points[i])) {**

**b.append("m ");**

**}**

**else {**

**b.append(points[i]);**

**b.append(' ');**

**}**

**}**

**return b.toString();**

**}**

**/\*\***

**\* Create a new Scribble object and initialize it by parsing a string of**

**\* coordinate data in the format produced by toString()**

**\*\*/**

**public static Scribble parse(String s) throws NumberFormatException {**

**StringTokenizer st = new StringTokenizer(s);**

**Scribble scribble = new Scribble();**

**while(st.hasMoreTokens()) {**

**String t = st.nextToken();**

**if (t.charAt(0) == 'm') {**

**scribble.moveto(Double.parseDouble(st.nextToken()),**

**Double.parseDouble(st.nextToken()));**

**}**

**else {**

**scribble.lineto(Double.parseDouble(t),**

**Double.parseDouble(st.nextToken()));**

**}**

**}**

**return scribble;**

**}**

**// ========= The following methods implement the Shape interface ========**

**/\*\* Return the bounding box of the Shape \*/**

**public Rectangle getBounds() {**

**return new Rectangle((int)(minX-0.5f), (int)(minY-0.5f),**

**(int)(maxX-minX+0.5f), (int)(maxY-minY+0.5f));**

**}**

**/\*\* Return the bounding box of the Shape \*/**

**public Rectangle2D getBounds2D() {**

**return new Rectangle2D.Double(minX, minY, maxX-minX, maxY-minY);**

**}**

**/\*\* Our shape is an open curve, so it never contains anything \*/**

**public boolean contains(Point2D p) { return false; }**

**public boolean contains(Rectangle2D r) { return false; }**

**public boolean contains(double x, double y) { return false; }**

**public boolean contains(double x, double y, double w, double h) {**

**return false;**

**}**

**/\*\***

**\* Determine if the scribble intersects the specified rectangle by testing**

**\* each line segment individually**

**\*\*/**

**public boolean intersects(Rectangle2D r) {**

**if (numPoints < 4) return false;**

**int i = 0;**

**double x1, y1, x2 = 0.0, y2 = 0.0;**

**while(i < numPoints) {**

**if (Double.isNaN(points[i])) { // If we're beginning a new line**

**i++; // Skip the NaN**

**x2 = points[i++];**

**y2 = points[i++];**

**}**

**else {**

**x1 = x2;**

**y1 = y2;**

**x2 = points[i++];**

**y2 = points[i++];**

**if (r.intersectsLine(x1, y1, x2, y2)) return true;**

**}**

**}**

**return false;**

**}**

**/\*\* Test for intersection by invoking the method above \*/**

**public boolean intersects(double x, double y, double w, double h){**

**return intersects(new Rectangle2D.Double(x,y,w,h));**

**}**

**/\*\***

**\* Return a PathIterator object that tells Java2D how to draw this scribble**

**\*\*/**

**public PathIterator getPathIterator(AffineTransform at) {**

**return new ScribbleIterator(at);**

**}**

**/\*\***

**\* Return a PathIterator that doesn't include curves. Ours never does.**

**\*\*/**

**public PathIterator getPathIterator(AffineTransform at, double flatness) {**

**return getPathIterator(at);**

**}**

**/\*\***

**\* This inner class implements the PathIterator interface to describe**

**\* the shape of a scribble. Since a Scribble is composed of arbitrary**

**\* movetos and linetos, we simply return their coordinates**

**\*\*/**

**public class ScribbleIterator implements PathIterator {**

**protected int i = 0; // Position in array**

**protected AffineTransform transform;**

**public ScribbleIterator(AffineTransform transform) {**

**this.transform = transform;**

**}**

**/\*\* How to determine insideness and outsideness for this shape \*/**

**public int getWindingRule() { return PathIterator.WIND\_NON\_ZERO; }**

**/\*\* Have we reached the end of the scribble path yet? \*/**

**public boolean isDone() { return i >= numPoints; }**

**/\*\* Move on to the next segment of the path \*/**

**public void next() {**

**if (Double.isNaN(points[i])) i += 3;**

**else i += 2;**

**}**

**/\*\***

**\* Get the coordinates of the current moveto or lineto as floats**

**\*\*/**

**public int currentSegment(float[] coords) {**

**int retval;**

**if (Double.isNaN(points[i])) { // If its a moveto**

**coords[0] = (float)points[i+1];**

**coords[1] = (float)points[i+2];**

**retval = SEG\_MOVETO;**

**}**

**else {**

**coords[0] = (float)points[i];**

**coords[1] = (float)points[i+1];**

**retval = SEG\_LINETO;**

**}**

**// If a transform was specified, use it on the coordinates**

**if (transform != null) transform.transform(coords, 0, coords, 0,1);**

**return retval;**

**}**

**/\*\***

**\* Get the coordinates of the current moveto or lineto as doubles**

**\*\*/**

**public int currentSegment(double[] coords) {**

**int retval;**

**if (Double.isNaN(points[i])) {**

**coords[0] = points[i+1];**

**coords[1] = points[i+2];**

**retval = SEG\_MOVETO;**

**}**

**else {**

**coords[0] = points[i];**

**coords[1] = points[i+1];**

**retval = SEG\_LINETO;**

**}**

**if (transform != null) transform.transform(coords, 0, coords, 0,1);**

**return retval;**

**}**

**}**

**//====== The following methods implement the Transferable interface =====**

**// This is the custom DataFlavor for Scribble objects**

**public static DataFlavor scribbleDataFlavor =**

**new DataFlavor(Scribble.class, "Scribble");**

**// This is a list of the flavors we know how to work with**

**public static DataFlavor[] supportedFlavors = {**

**scribbleDataFlavor,**

**DataFlavor.stringFlavor**

**};**

**/\*\* Return the data formats or "flavors" we know how to transfer \*/**

**public DataFlavor[] getTransferDataFlavors() {**

**return (DataFlavor[]) supportedFlavors.clone();**

**}**

**/\*\* Check whether we support a given flavor \*/**

**public boolean isDataFlavorSupported(DataFlavor flavor) {**

**return (flavor.equals(scribbleDataFlavor) ||**

**flavor.equals(DataFlavor.stringFlavor));**

**}**

**/\*\***

**\* Return the scribble data in the requested format, or throw an exception**

**\* if we don't support the requested format**

**\*\*/**

**public Object getTransferData(DataFlavor flavor)**

**throws UnsupportedFlavorException**

**{**

**if (flavor.equals(scribbleDataFlavor)) { return this; }**

**else if (flavor.equals(DataFlavor.stringFlavor)) { return toString(); }**

**else throw new UnsupportedFlavorException(flavor);**

**}**

**}**

**ScribbleDragAngDrop.java**

**import java.awt.\*;**

**import java.awt.event.\*;**

**import javax.swing.\*;**

**import javax.swing.border.\*;**

**import java.awt.datatransfer.\*; // Clipboard, Transferable, DataFlavor, etc.**

**import java.awt.dnd.\*;**

**import java.util.ArrayList;**

**/\*\***

**\* This component can operate in two modes. In "draw mode", it allows the user**

**\* to scribble with the mouse. In "drag mode", it allows the user to drag**

**\* scribbles with the mouse. Regardless of the mode, it always allows**

**\* scribbles to be dropped on it from other applications.**

**\*\*/**

**public class ScribbleDragAndDrop extends JComponent**

**implements DragGestureListener, // For recognizing the start of drags**

**DragSourceListener, // For processing drag source events**

**DropTargetListener, // For processing drop target events**

**MouseListener, // For processing mouse clicks**

**MouseMotionListener // For processing mouse drags**

**{**

**ArrayList<Scribble> scribbles = new ArrayList<Scribble>();**

**Stroke stroke = new MyStroke(2.0f, 8.0f, 8.0f);**

**// A list of Scribbles to draw**

**Scribble currentScribble; // The scribble in progress**

**Scribble beingDragged; // The scribble being dragged**

**DragSource dragSource; // A central DnD object**

**boolean dragMode; // Are we dragging or scribbling?**

**// These are some constants we use**

**static final int LINEWIDTH = 3;**

**static final BasicStroke linestyle = new BasicStroke(LINEWIDTH);**

**static final Border normalBorder = new BevelBorder(BevelBorder.LOWERED);**

**static final Border dropBorder = new BevelBorder(BevelBorder.RAISED);**

**/\*\* The constructor: set up drag-and-drop stuff \*/**

**public ScribbleDragAndDrop() {**

**// Give ourselves a nice default border.**

**// We'll change this border during drag-and-drop.**

**setBorder(normalBorder);**

**// Register listeners to handle drawing**

**addMouseListener(this);**

**addMouseMotionListener(this);**

**// Create a DragSource and DragGestureRecognizer to listen for drags**

**// The DragGestureRecognizer will notify the DragGestureListener**

**// when the user tries to drag an object**

**dragSource = DragSource.getDefaultDragSource();**

**dragSource.createDefaultDragGestureRecognizer(this, // What component**

**DnDConstants.ACTION\_COPY\_OR\_MOVE, // What drag types?**

**this);// the listener**

**// Create and set up a DropTarget that will listen for drags and**

**// drops over this component, and will notify the DropTargetListener**

**DropTarget dropTarget = new DropTarget(this, // component to monitor**

**this); // listener to notify**

**this.setDropTarget(dropTarget); // Tell the component about it.**

**}**

**/\*\***

**\* The component draws itself by drawing each of the Scribble objects.**

**\*\*/**

**public void paintComponent(Graphics g) {**

**super.paintComponent(g);**

**Graphics2D g2 = (Graphics2D) g;**

**g2.setStroke(linestyle); // Specify wide lines**

**int numScribbles = scribbles.size();**

**g2.setStroke(stroke);**

**for(int i = 0; i < numScribbles; i++) {**

**Scribble s = (Scribble)scribbles.get(i);**

**g2.draw(s); // Draw the scribble**

**}**

**g2.setStroke(linestyle);**

**}**

**public void setDragMode(boolean dragMode) {**

**this.dragMode = dragMode;**

**}**

**public boolean getDragMode() { return dragMode; }**

**/\*\***

**\* This method, and the following four methods are from the MouseListener**

**\* interface. If we're in drawing mode, this method handles mouse down**

**\* events and starts a new scribble.**

**\*\*/**

**public void mousePressed(MouseEvent e) {**

**if (dragMode) return;**

**currentScribble = new Scribble();**

**scribbles.add(currentScribble);**

**currentScribble.draw(e.getX()-100, e.getY()-100, 200, 200);**

**repaint();**

**}**

**public void mouseReleased(MouseEvent e) {}**

**public void mouseClicked(MouseEvent e) {}**

**public void mouseEntered(MouseEvent e) {}**

**public void mouseExited(MouseEvent e) {}**

**/\*\***

**\* This method and mouseMoved() below are from the MouseMotionListener**

**\* interface. If we're in drawing mode, this method adds a new point**

**\* to the current scribble and requests a redraw**

**\*\*/**

**public void mouseDragged(MouseEvent e) {**

**/\*if (dragMode) return;**

**currentScribble.lineto(e.getX(), e.getY());**

**repaint();\*/**

**}**

**public void mouseMoved(MouseEvent e) {}**

**/\*\***

**\* This method implements the DragGestureListener interface. It will be**

**\* invoked when the DragGestureRecognizer thinks that the user has**

**\* initiated a drag. If we're not in drawing mode, then this method will**

**\* try to figure out which Scribble object is being dragged, and will**

**\* initiate a drag on that object.**

**\*\*/**

**public void dragGestureRecognized(DragGestureEvent e) {**

**// Don't drag if we're not in drag mode**

**if (!dragMode) return;**

**// Figure out where the drag started**

**MouseEvent inputEvent = (MouseEvent) e.getTriggerEvent();**

**int x = inputEvent.getX();**

**int y = inputEvent.getY();**

**// Figure out which scribble was clicked on, if any by creating a**

**// small rectangle around the point and testing for intersection.**

**Rectangle r = new Rectangle (x-LINEWIDTH, y-LINEWIDTH,**

**LINEWIDTH\*2, LINEWIDTH\*2);**

**int numScribbles = scribbles.size();**

**for(int i = 0; i < numScribbles; i++) { // Loop through the scribbles**

**Scribble s = (Scribble) scribbles.get(i);**

**if (s.intersects(r)) {**

**// The user started the drag on top of this scribble, so**

**// start to drag it.**

**// First, remember which scribble is being dragged, so we can**

**// delete it later (if this is a move rather than a copy)**

**beingDragged = s;**

**// Next, create a copy that will be the one dragged**

**Scribble dragScribble = (Scribble) s.clone();**

**// Adjust the origin to the point the user clicked on.**

**dragScribble.translate(-x, -y);**

**// Choose a cursor based on the type of drag the user initiated**

**Cursor cursor;**

**switch(e.getDragAction()) {**

**case DnDConstants.ACTION\_COPY:**

**cursor = DragSource.DefaultCopyDrop;**

**break;**

**case DnDConstants.ACTION\_MOVE:**

**cursor = DragSource.DefaultMoveDrop;**

**break;**

**default:**

**return; // We only support move and copys**

**}**

**// Some systems allow us to drag an image along with the**

**// cursor. If so, create an image of the scribble to drag**

**if (DragSource.isDragImageSupported()) {**

**Rectangle scribbleBox = dragScribble.getBounds();**

**Image dragImage = this.createImage(scribbleBox.width,**

**scribbleBox.height);**

**Graphics2D g = (Graphics2D)dragImage.getGraphics();**

**g.setColor(new Color(0,0,0,0)); // transparent background**

**g.fillRect(0, 0, scribbleBox.width, scribbleBox.height);**

**g.setColor(Color.black);**

**g.setStroke(linestyle);**

**g.translate(-scribbleBox.x, -scribbleBox.y);**

**g.draw(dragScribble);**

**Point hotspot = new Point(-scribbleBox.x, -scribbleBox.y);**

**// Now start dragging, using the image.**

**e.startDrag(cursor, dragImage, hotspot, dragScribble,this);**

**}**

**else {**

**// Or start the drag without an image**

**e.startDrag(cursor, dragScribble,this);**

**}**

**// After we've started dragging one scribble, stop looking**

**return;**

**}**

**}**

**}**

**/\*\***

**\* This method, and the four unused methods that follow it implement the**

**\* DragSourceListener interface. dragDropEnd() is invoked when the user**

**\* drops the scribble she was dragging. If the drop was successful, and**

**\* if the user did a "move" rather than a "copy", then we delete the**

**\* dragged scribble from the list of scribbles to draw.**

**\*\*/**

**public void dragDropEnd(DragSourceDropEvent e) {**

**if (!e.getDropSuccess()) return;**

**int action = e.getDropAction();**

**if (action == DnDConstants.ACTION\_MOVE) {**

**scribbles.remove(beingDragged);**

**beingDragged = null;**

**repaint();**

**}**

**}**

**// These methods are also part of DragSourceListener.**

**// They are invoked at interesting points during the drag, and can be**

**// used to perform "drag over" effects, such as changing the drag cursor**

**// or drag image.**

**public void dragEnter(DragSourceDragEvent e) {}**

**public void dragExit(DragSourceEvent e) {}**

**public void dropActionChanged(DragSourceDragEvent e) {}**

**public void dragOver(DragSourceDragEvent e) {}**

**// The next five methods implement DropTargetListener**

**/\*\***

**\* This method is invoked when the user first drags something over us.**

**\* If we understand the data type being dragged, then call acceptDrag()**

**\* to tell the system that we're receptive. Also, we change our border**

**\* as a "drag under" effect to signal that we can accept the drop.**

**\*\*/**

**public void dragEnter(DropTargetDragEvent e) {**

**if (e.isDataFlavorSupported(Scribble.scribbleDataFlavor) ||**

**e.isDataFlavorSupported(DataFlavor.stringFlavor)) {**

**e.acceptDrag(DnDConstants.ACTION\_COPY\_OR\_MOVE);**

**this.setBorder(dropBorder);**

**}**

**}**

**/\*\* The user is no longer dragging over us, so restore the border \*/**

**public void dragExit(DropTargetEvent e) { this.setBorder(normalBorder); }**

**/\*\***

**\* This is the key method of DropTargetListener. It is invoked when the**

**\* user drops something on us.**

**\*\*/**

**public void drop(DropTargetDropEvent e) {**

**this.setBorder(normalBorder); // Restore the default border**

**// First, check whether we understand the data that was dropped.**

**// If we supports our data flavors, accept the drop, otherwise reject.**

**if (e.isDataFlavorSupported(Scribble.scribbleDataFlavor) ||**

**e.isDataFlavorSupported(DataFlavor.stringFlavor)) {**

**e.acceptDrop(DnDConstants.ACTION\_COPY\_OR\_MOVE);**

**}**

**else {**

**e.rejectDrop();**

**return;**

**}**

**// We've accepted the drop, so now we attempt to get the dropped data**

**// from the Transferable object.**

**Transferable t = e.getTransferable(); // Holds the dropped data**

**Scribble droppedScribble; // This will hold the Scribble object**

**// First, try to get the data directly as a scribble object**

**try {**

**droppedScribble =**

**(Scribble) t.getTransferData(Scribble.scribbleDataFlavor);**

**}**

**catch (Exception ex) { // unsupported flavor, IO exception, etc.**

**// If that doesn't work, try to get it as a String and parse it**

**try {**

**String s = (String) t.getTransferData(DataFlavor.stringFlavor);**

**droppedScribble = Scribble.parse(s);**

**}**

**catch(Exception ex2) {**

**// If we still couldn't get the data, tell the system we failed**

**e.dropComplete(false);**

**return;**

**}**

**}**

**// If we get here, we've got the Scribble object**

**Point p = e.getLocation(); // Where did the drop happen?**

**droppedScribble.translate(p.getX(), p.getY()); // Move it there**

**scribbles.add(droppedScribble); // add to display list**

**repaint(); // ask for redraw**

**e.dropComplete(true); // signal success!**

**}**

**// These are unused DropTargetListener methods**

**public void dragOver(DropTargetDragEvent e) {}**

**public void dropActionChanged(DropTargetDragEvent e) {}**

**/\*\***

**\* The main method. Creates a simple application using this class. Note**

**\* the buttons for switching between draw mode and drag mode.**

**\*\*/**

**public static void main(String[] args) {**

**// Create a frame and put a scribble pane in it**

**JFrame frame = new JFrame("ScribbleDragAndDrop");**

**frame.addWindowListener(new WindowAdapter() {**

**public void windowClosing(WindowEvent e) { System.exit(0); }**

**});**

**final ScribbleDragAndDrop scribblePane = new ScribbleDragAndDrop();**

**frame.getContentPane().add(scribblePane, BorderLayout.CENTER);**

**// Create two buttons for switching modes**

**JToolBar toolbar = new JToolBar();**

**ButtonGroup group = new ButtonGroup();**

**JToggleButton draw = new JToggleButton("Draw");**

**JToggleButton drag = new JToggleButton("Drag");**

**draw.addActionListener(new ActionListener() {**

**public void actionPerformed(ActionEvent e) {**

**scribblePane.setDragMode(false);**

**}**

**});**

**drag.addActionListener(new ActionListener() {**

**public void actionPerformed(ActionEvent e) {**

**scribblePane.setDragMode(true);**

**}**

**});**

**group.add(draw); group.add(drag);**

**toolbar.add(draw); toolbar.add(drag);**

**frame.getContentPane().add(toolbar, BorderLayout.NORTH);**

**// Start off in drawing mode**

**draw.setSelected(true);**

**scribblePane.setDragMode(false);**

**// Pop up the window**

**frame.setSize(400, 400);**

**frame.setVisible(true);**

**}**

**}**

**A.6 Исходные коды лабораторной работы №6**

**Main.java**

**import java.awt.Container;**

**import java.awt.GridLayout;**

**import java.util.ArrayList;**

**import java.util.List;**

**import javax.swing.JFrame;**

**import javax.swing.JScrollPane;**

**import javax.swing.SwingUtilities;**

**import javax.swing.UIManager;**

**public class Main extends JFrame {**

**private static final long serialVersionUID = 1L;**

**public Main() {**

**super("MyTreeTable");**

**setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);**

**setLayout(new GridLayout(0, 1));**

**MyAbstractTreeTableModel treeTableModel = new MyDataModel(createDataStructure());**

**MyTreeTable myTreeTable = new MyTreeTable(treeTableModel);**

**Container cPane = getContentPane();**

**cPane.add(new JScrollPane(myTreeTable));**

**setSize(800, 600);**

**setLocationRelativeTo(null);**

**}**

**private static MyDataNode createDataStructure() {**

**List<MyDataNode> brest = new ArrayList<MyDataNode>();**

**brest.add(new MyDataNode("Эконом","06:34","07:23","Ежедневно", null));**

**brest.add(new MyDataNode("Эконом","07:24","08:45","Ежедневно", null));**

**brest.add(new MyDataNode("Экспесс","08:13","09:23","По выходным", null));**

**brest.add(new MyDataNode("Экспесс","09:45","10:55","Ежедневно", null));**

**brest.add(new MyDataNode("Эконом","10:33","11:23","Ежедневно", null));**

**brest.add(new MyDataNode("Экспесс","11:34","12:43","По выходным", null));**

**brest.add(new MyDataNode("Экспесс","12:12","13:03","Ежедневно", null));**

**brest.add(new MyDataNode("Эконом","13:43","14:54","Ежедневно", null));**

**brest.add(new MyDataNode("Экспесс","14:21","15:34","По выходным", null));**

**brest.add(new MyDataNode("Эконом","15:21","16:21","Ежедневно", null));**

**brest.add(new MyDataNode("Эконом","16:34","17:19","Ежедневно", null));**

**brest.add(new MyDataNode("Экспесс","17:01","18:22","По выходным", null));**

**brest.add(new MyDataNode("Эконом","18:04","19:05","Ежедневно", null));**

**brest.add(new MyDataNode("Эконом","19:34","20:06","Ежедневно", null));**

**brest.add(new MyDataNode("Экспесс","20:57","21:44","По выходным", null));**

**brest.add(new MyDataNode("Экспесс","21:34","22:25","Ежедневно", null));**

**brest.add(new MyDataNode("Эконом","22:09","23:07","Ежедневно", null));**

**brest.add(new MyDataNode("Экспесс","23:00","00:00","По выходным", null));**

**List<MyDataNode> grodno = new ArrayList<MyDataNode>();**

**grodno.add(new MyDataNode("Эконом","06:34","07:23","Ежедневно", null));**

**grodno.add(new MyDataNode("Эконом","07:24","08:45","Ежедневно", null));**

**grodno.add(new MyDataNode("Экспесс","08:13","09:23","По выходным", null));**

**grodno.add(new MyDataNode("Экспесс","09:45","10:55","Ежедневно", null));**

**grodno.add(new MyDataNode("Эконом","10:33","11:23","Ежедневно", null));**

**grodno.add(new MyDataNode("Экспесс","11:34","12:43","По выходным", null));**

**grodno.add(new MyDataNode("Экспесс","12:12","13:03","Ежедневно", null));**

**grodno.add(new MyDataNode("Эконом","13:43","14:54","Ежедневно", null));**

**grodno.add(new MyDataNode("Экспесс","14:21","15:34","По выходным", null));**

**grodno.add(new MyDataNode("Эконом","15:21","16:21","Ежедневно", null));**

**grodno.add(new MyDataNode("Эконом","16:34","17:19","Ежедневно", null));**

**grodno.add(new MyDataNode("Экспесс","17:01","18:22","По выходным", null));**

**grodno.add(new MyDataNode("Эконом","18:04","19:05","Ежедневно", null));**

**grodno.add(new MyDataNode("Эконом","19:34","20:06","Ежедневно", null));**

**grodno.add(new MyDataNode("Экспесс","20:57","21:44","По выходным", null));**

**grodno.add(new MyDataNode("Экспесс","21:34","22:25","Ежедневно", null));**

**grodno.add(new MyDataNode("Эконом","22:09","23:07","Ежедневно", null));**

**grodno.add(new MyDataNode("Экспесс","23:00","00:00","По выходным", null));**

**List<MyDataNode> witebsk = new ArrayList<MyDataNode>();**

**witebsk.add(new MyDataNode("Эконом","06:34","07:23","Ежедневно", null));**

**witebsk.add(new MyDataNode("Эконом","07:24","08:45","Ежедневно", null));**

**witebsk.add(new MyDataNode("Экспесс","08:13","09:23","По выходным", null));**

**witebsk.add(new MyDataNode("Экспесс","09:45","10:55","Ежедневно", null));**

**witebsk.add(new MyDataNode("Эконом","10:33","11:23","Ежедневно", null));**

**witebsk.add(new MyDataNode("Экспесс","11:34","12:43","По выходным", null));**

**witebsk.add(new MyDataNode("Экспесс","12:12","13:03","Ежедневно", null));**

**witebsk.add(new MyDataNode("Эконом","13:43","14:54","Ежедневно", null));**

**witebsk.add(new MyDataNode("Экспесс","14:21","15:34","По выходным", null));**

**witebsk.add(new MyDataNode("Эконом","15:21","16:21","Ежедневно", null));**

**witebsk.add(new MyDataNode("Эконом","16:34","17:19","Ежедневно", null));**

**witebsk.add(new MyDataNode("Экспесс","17:01","18:22","По выходным", null));**

**witebsk.add(new MyDataNode("Эконом","18:04","19:05","Ежедневно", null));**

**witebsk.add(new MyDataNode("Эконом","19:34","20:06","Ежедневно", null));**

**witebsk.add(new MyDataNode("Экспесс","20:57","21:44","По выходным", null));**

**witebsk.add(new MyDataNode("Экспесс","21:34","22:25","Ежедневно", null));**

**witebsk.add(new MyDataNode("Эконом","22:09","23:07","Ежедневно", null));**

**witebsk.add(new MyDataNode("Экспесс","23:00","00:00","По выходным", null));**

**List<MyDataNode> gomel = new ArrayList<MyDataNode>();**

**gomel.add(new MyDataNode("Эконом","06:34","07:23","Ежедневно", null));**

**gomel.add(new MyDataNode("Эконом","07:24","08:45","Ежедневно", null));**

**gomel.add(new MyDataNode("Экспесс","08:13","09:23","По выходным", null));**

**gomel.add(new MyDataNode("Экспесс","09:45","10:55","Ежедневно", null));**

**gomel.add(new MyDataNode("Эконом","10:33","11:23","Ежедневно", null));**

**gomel.add(new MyDataNode("Экспесс","11:34","12:43","По выходным", null));**

**gomel.add(new MyDataNode("Экспесс","12:12","13:03","Ежедневно", null));**

**gomel.add(new MyDataNode("Эконом","13:43","14:54","Ежедневно", null));**

**gomel.add(new MyDataNode("Экспесс","14:21","15:34","По выходным", null));**

**gomel.add(new MyDataNode("Эконом","15:21","16:21","Ежедневно", null));**

**gomel.add(new MyDataNode("Эконом","16:34","17:19","Ежедневно", null));**

**gomel.add(new MyDataNode("Экспесс","17:01","18:22","По выходным", null));**

**gomel.add(new MyDataNode("Эконом","18:04","19:05","Ежедневно", null));**

**gomel.add(new MyDataNode("Эконом","19:34","20:06","Ежедневно", null));**

**gomel.add(new MyDataNode("Экспесс","20:57","21:44","По выходным", null));**

**gomel.add(new MyDataNode("Экспесс","21:34","22:25","Ежедневно", null));**

**gomel.add(new MyDataNode("Эконом","22:09","23:07","Ежедневно", null));**

**gomel.add(new MyDataNode("Экспесс","23:00","00:00","По выходным", null));**

**List<MyDataNode> rootNodes = new ArrayList<MyDataNode>();**

**rootNodes.add(new MyDataNode("Брест","","","",brest));**

**rootNodes.add(new MyDataNode("Гродно","","","",grodno));**

**rootNodes.add(new MyDataNode("Витебск","","","",witebsk));**

**rootNodes.add(new MyDataNode("Гомель","","","",gomel));**

**MyDataNode root = new MyDataNode("Минск-", "", "", "", rootNodes);**

**return root;**

**}**

**public static void main(final String[] args) {**

**Runnable gui = new Runnable() {**

**public void run() {**

**try {**

**UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());**

**} catch (Exception e) {**

**e.printStackTrace();**

**}**

**new Main().setVisible(true);**

**}**

**};**

**SwingUtilities.invokeLater(gui);**

**}**

**}**

**MyTreeTable.java**

**import java.awt.Dimension;**

**import javax.swing.JTable;**

**import java.awt.Component;**

**import java.awt.Graphics;**

**import javax.swing.JTree;**

**import javax.swing.table.TableCellRenderer;**

**import javax.swing.tree.TreeModel;**

**import javax.swing.event.TreeExpansionEvent;**

**import javax.swing.event.TreeExpansionListener;**

**import javax.swing.table.AbstractTableModel;**

**import javax.swing.tree.TreePath;**

**import java.awt.event.MouseEvent;**

**import java.util.EventObject;**

**import javax.swing.AbstractCellEditor;**

**import javax.swing.table.TableCellEditor;**

**import javax.swing.ListSelectionModel;**

**import javax.swing.event.ListSelectionEvent;**

**import javax.swing.event.ListSelectionListener;**

**import javax.swing.tree.DefaultTreeSelectionModel;**

**import java.util.Collections;**

**import java.util.List;**

**import javax.swing.event.EventListenerList;**

**import javax.swing.event.TreeModelEvent;**

**import javax.swing.event.TreeModelListener;**

**public class MyTreeTable extends JTable {**

**private static final long serialVersionUID = 1L;**

**private MyTreeTableCellRenderer tree;**

**public MyTreeTable(MyAbstractTreeTableModel treeTableModel) {**

**super();**

**tree = new MyTreeTableCellRenderer(this, treeTableModel);**

**super.setModel(new MyTreeTableModelAdapter(treeTableModel, tree));**

**MyTreeTableSelectionModel selectionModel = new MyTreeTableSelectionModel();**

**tree.setSelectionModel(selectionModel); // For the tree**

**setSelectionModel(selectionModel.getListSelectionModel()); // For the**

**setDefaultRenderer(MyTreeTableModel.class, tree);**

**setDefaultEditor(MyTreeTableModel.class, new MyTreeTableCellEditor(**

**tree, this));**

**setShowGrid(false);**

**setIntercellSpacing(new Dimension(0, 0));**

**}**

**}**

**class MyTreeTableCellRenderer extends JTree implements TableCellRenderer {**

**private static final long serialVersionUID = 1L;**

**protected int visibleRow;**

**private MyTreeTable treeTable;**

**public MyTreeTableCellRenderer(MyTreeTable treeTable, TreeModel model) {**

**super(model);**

**this.treeTable = treeTable;**

**setRowHeight(getRowHeight());**

**}**

**public void setRowHeight(int rowHeight) {**

**if (rowHeight > 0) {**

**super.setRowHeight(rowHeight);**

**if (treeTable != null && treeTable.getRowHeight() != rowHeight) {**

**treeTable.setRowHeight(getRowHeight());**

**}**

**}**

**}**

**public void setBounds(int x, int y, int w, int h) {**

**super.setBounds(x, 0, w, treeTable.getHeight());**

**}**

**public void paint(Graphics g) {**

**g.translate(0, -visibleRow \* getRowHeight());**

**super.paint(g);**

**}**

**public Component getTableCellRendererComponent(JTable table, Object value,**

**boolean isSelected, boolean hasFocus, int row, int column) {**

**if (isSelected)**

**setBackground(table.getSelectionBackground());**

**else**

**setBackground(table.getBackground());**

**visibleRow = row;**

**return this;**

**}**

**}**

**class MyTreeTableModelAdapter extends AbstractTableModel {**

**private static final long serialVersionUID = 1L;**

**JTree tree;**

**MyAbstractTreeTableModel treeTableModel;**

**public MyTreeTableModelAdapter(MyAbstractTreeTableModel treeTableModel,**

**JTree tree) {**

**this.tree = tree;**

**this.treeTableModel = treeTableModel;**

**tree.addTreeExpansionListener(new TreeExpansionListener() {**

**public void treeExpanded(TreeExpansionEvent event) {**

**fireTableDataChanged();**

**}**

**public void treeCollapsed(TreeExpansionEvent event) {**

**fireTableDataChanged();**

**}**

**});**

**}**

**public int getColumnCount() {**

**return treeTableModel.getColumnCount();**

**}**

**public String getColumnName(int column) {**

**return treeTableModel.getColumnName(column);**

**}**

**public Class<?> getColumnClass(int column) {**

**return treeTableModel.getColumnClass(column);**

**}**

**public int getRowCount() {**

**return tree.getRowCount();**

**}**

**protected Object nodeForRow(int row) {**

**TreePath treePath = tree.getPathForRow(row);**

**return treePath.getLastPathComponent();**

**}**

**public Object getValueAt(int row, int column) {**

**return treeTableModel.getValueAt(nodeForRow(row), column);**

**}**

**public boolean isCellEditable(int row, int column) {**

**return treeTableModel.isCellEditable(nodeForRow(row), column);**

**}**

**public void setValueAt(Object value, int row, int column) {**

**treeTableModel.setValueAt(value, nodeForRow(row), column);**

**}**

**}**

**class MyTreeTableCellEditor extends AbstractCellEditor implements**

**TableCellEditor {**

**private static final long serialVersionUID = 1L;**

**private JTree tree;**

**private JTable table;**

**public MyTreeTableCellEditor(JTree tree, JTable table) {**

**this.tree = tree;**

**this.table = table;**

**}**

**public Component getTableCellEditorComponent(JTable table, Object value,**

**boolean isSelected, int r, int c) {**

**return tree;**

**}**

**public boolean isCellEditable(EventObject e) {**

**if (e instanceof MouseEvent) {**

**int colunm1 = 0;**

**MouseEvent me = (MouseEvent) e;**

**int doubleClick = 2;**

**MouseEvent newME = new MouseEvent(tree, me.getID(), me.getWhen(),**

**me.getModifiers(), me.getX()**

**- table.getCellRect(0, colunm1, true).x, me.getY(),**

**doubleClick, me.isPopupTrigger());**

**tree.dispatchEvent(newME);**

**}**

**return false;**

**}**

**@Override**

**public Object getCellEditorValue() {**

**return null;**

**}**

**}**

**class MyTreeTableSelectionModel extends DefaultTreeSelectionModel {**

**private static final long serialVersionUID = 1L;**

**public MyTreeTableSelectionModel() {**

**super();**

**getListSelectionModel().addListSelectionListener(**

**new ListSelectionListener() {**

**@Override**

**public void valueChanged(ListSelectionEvent e) {**

**}**

**});**

**}**

**ListSelectionModel getListSelectionModel() {**

**return listSelectionModel;**

**}**

**}**

**class MyDataNode {**

**private String name;**

**private String processor;**

**private String ram;**

**private String disc;**

**private List<MyDataNode> children;**

**public MyDataNode(String name, String processor, String ram, String disc,**

**List<MyDataNode> children) {**

**this.name = name;**

**this.processor = processor;**

**this.ram = ram;**

**this.disc = disc;**

**this.children = children;**

**if (this.children == null) {**

**this.children = Collections.emptyList();**

**}**

**}**

**public String getName() {**

**return name;**

**}**

**public String getProcessor() {**

**return processor;**

**}**

**public String getRam() {**

**return ram;**

**}**

**public String getDisc() {**

**return disc;**

**}**

**public List<MyDataNode> getChildren() {**

**return children;**

**}**

**public String toString() {**

**return name;**

**}**

**}**

**class MyDataModel extends MyAbstractTreeTableModel {**

**// Spalten Name.**

**static protected String[] columnNames = { "Направление", "Время отъезда", "Время приезда",**

**"По дням" };**

**// Spalten Typen.**

**static protected Class<?>[] columnTypes = { MyTreeTableModel.class,**

**String.class, String.class, String.class };**

**public MyDataModel(MyDataNode rootNode) {**

**super(rootNode);**

**root = rootNode;**

**}**

**public Object getChild(Object parent, int index) {**

**return ((MyDataNode) parent).getChildren().get(index);**

**}**

**public int getChildCount(Object parent) {**

**return ((MyDataNode) parent).getChildren().size();**

**}**

**public int getColumnCount() {**

**return columnNames.length;**

**}**

**public String getColumnName(int column) {**

**return columnNames[column];**

**}**

**public Class<?> getColumnClass(int column) {**

**return columnTypes[column];**

**}**

**public Object getValueAt(Object node, int column) {**

**switch (column) {**

**case 0:**

**return ((MyDataNode) node).getName();**

**case 1:**

**return ((MyDataNode) node).getProcessor();**

**case 2:**

**return ((MyDataNode) node).getRam();**

**case 3:**

**return ((MyDataNode) node).getDisc();**

**default:**

**break;**

**}**

**return null;**

**}**

**public boolean isCellEditable(Object node, int column) {**

**return true; // Important to activate TreeExpandListener**

**}**

**public void setValueAt(Object aValue, Object node, int column) {**

**}**

**}**

**abstract class MyAbstractTreeTableModel implements MyTreeTableModel {**

**protected Object root;**

**protected EventListenerList listenerList = new EventListenerList();**

**private static final int CHANGED = 0;**

**private static final int INSERTED = 1;**

**private static final int REMOVED = 2;**

**private static final int STRUCTURE\_CHANGED = 3;**

**public MyAbstractTreeTableModel(Object root) {**

**this.root = root;**

**}**

**public Object getRoot() {**

**return root;**

**}**

**public boolean isLeaf(Object node) {**

**return getChildCount(node) == 0;**

**}**

**public void valueForPathChanged(TreePath path, Object newValue) {**

**}**

**public int getIndexOfChild(Object parent, Object child) {**

**return 0;**

**}**

**public void addTreeModelListener(TreeModelListener l) {**

**listenerList.add(TreeModelListener.class, l);**

**}**

**public void removeTreeModelListener(TreeModelListener l) {**

**listenerList.remove(TreeModelListener.class, l);**

**}**

**private void fireTreeNode(int changeType, Object source, Object[] path,**

**int[] childIndices, Object[] children) {**

**Object[] listeners = listenerList.getListenerList();**

**TreeModelEvent e = new TreeModelEvent(source, path, childIndices,**

**children);**

**for (int i = listeners.length - 2; i >= 0; i -= 2) {**

**if (listeners[i] == TreeModelListener.class) {**

**switch (changeType) {**

**case CHANGED:**

**((TreeModelListener) listeners[i + 1]).treeNodesChanged(e);**

**break;**

**case INSERTED:**

**((TreeModelListener) listeners[i + 1]).treeNodesInserted(e);**

**break;**

**case REMOVED:**

**((TreeModelListener) listeners[i + 1]).treeNodesRemoved(e);**

**break;**

**case STRUCTURE\_CHANGED:**

**((TreeModelListener) listeners[i + 1])**

**.treeStructureChanged(e);**

**break;**

**default:**

**break;**

**}**

**}**

**}**

**}**

**protected void fireTreeNodesChanged(Object source, Object[] path,**

**int[] childIndices, Object[] children) {**

**fireTreeNode(CHANGED, source, path, childIndices, children);**

**}**

**protected void fireTreeNodesInserted(Object source, Object[] path,**

**int[] childIndices, Object[] children) {**

**fireTreeNode(INSERTED, source, path, childIndices, children);**

**}**

**protected void fireTreeNodesRemoved(Object source, Object[] path,**

**int[] childIndices, Object[] children) {**

**fireTreeNode(REMOVED, source, path, childIndices, children);**

**}**

**protected void fireTreeStructureChanged(Object source, Object[] path,**

**int[] childIndices, Object[] children) {**

**fireTreeNode(STRUCTURE\_CHANGED, source, path, childIndices, children);**

**}**

**}**

**interface MyTreeTableModel extends TreeModel {**

**public int getColumnCount();**

**public String getColumnName(int column);**

**public Class<?> getColumnClass(int column);**

**public Object getValueAt(Object node, int column);**

**public boolean isCellEditable(Object node, int column);**

**public void setValueAt(Object aValue, Object node, int column);**

**}**

**A.7 Исходные коды лабораторной работы №7**

**Alilnment.java**

**/\*\* This class defines an enumerated type with three values \*/**

**public class Alignment {**

**/\*\* This private constructor prevents anyone from instantiating us \*/**

**private Alignment() {};**

**// The following three constants are the only instances of this class**

**public static final Alignment *LEFT* = new Alignment();**

**public static final Alignment *CENTER* = new Alignment();**

**public static final Alignment *RIGHT* = new Alignment();**

**}**

**MultiLineLabel.java**

**import java.awt.\*;**

**import java.util.\*;**

**/\*\***

**\* A custom component that displays multiple lines of text with specified**

**\* margins and alignment. We could also subclass Component, making this**

**\* a "lightweight" component.**

**\*\*/**

**public class MultiLineLabel extends Canvas {**

**// User-specified properties**

**protected String label; // The label, not broken into lines**

**protected int margin\_width; // Left and right margins**

**protected int margin\_height; // Top and bottom margins**

**protected Alignment alignment; // The alignment of the text.**

**// Computed state values**

**protected int num\_lines; // The number of lines**

**protected String[] lines; // The label, broken into lines**

**protected int[] line\_widths; // How wide each line is**

**protected int max\_width; // The width of the widest line**

**protected int line\_height; // Total height of the font**

**protected int line\_ascent; // Font height above baseline**

**protected boolean measured = false; // Have the lines been measured?**

**// Here are five versions of the constructor.**

**public MultiLineLabel(String label, int margin\_width,**

**int margin\_height, Alignment alignment) {**

**this.label = label; // Remember all the properties.**

**this.margin\_width = margin\_width;**

**this.margin\_height = margin\_height;**

**this.alignment = alignment;**

**newLabel(); // Break the label up into lines.**

**}**

**public MultiLineLabel(String label, int margin\_width, int margin\_height) {**

**this(label, margin\_width, margin\_height, Alignment.LEFT);**

**}**

**public MultiLineLabel(String label, Alignment alignment) {**

**this(label, 10, 10, alignment);**

**}**

**public MultiLineLabel(String label) { this(label, 10, 10, Alignment.LEFT);}**

**public MultiLineLabel() { this(""); }**

**// Methods to set and query the various attributes of the component.**

**// Note that some query methods are inherited from the superclass.**

**public void setLabel(String label) {**

**this.label = label;**

**newLabel(); // Break the label into lines.**

**measured = false; // Note that we need to measure lines.**

**repaint(); // Request a redraw.**

**}**

**public void setFont(Font f) {**

**super.setFont(f); // Tell our superclass about the new font.**

**measured = false; // Note that we need to remeasure lines.**

**repaint(); // Request a redraw.**

**}**

**public void setForeground(Color c) {**

**super.setForeground(c); // Tell our superclass about the new color.**

**repaint(); // Request a redraw (size is unchanged).**

**}**

**public void setAlignment(Alignment a) { alignment = a; repaint(); }**

**public void setMarginWidth(int mw) { margin\_width = mw; repaint(); }**

**public void setMarginHeight(int mh) { margin\_height = mh; repaint(); }**

**// Property getter methods. Note that getFont(), getForeground(), etc.**

**// are inherited from the superclass.**

**public String getLabel() { return label; }**

**public Alignment getAlignment() { return alignment; }**

**public int getMarginWidth() { return margin\_width; }**

**public int getMarginHeight() { return margin\_height; }**

**/\*\***

**\* This method is called by a layout manager when it wants to**

**\* know how big we'd like to be. In Java 1.1, getPreferredSize() is**

**\* the preferred version of this method. We use this deprecated version**

**\* so that this component can interoperate with 1.0 components.**

**\*/**

**public Dimension getPreferredSize() {**

**if (!measured) measure();**

**return new Dimension(max\_width + 2\*margin\_width,**

**num\_lines \* line\_height + 2\*margin\_height);**

**}**

**/\*\***

**\* This method is called when the layout manager wants to know**

**\* the bare minimum amount of space we need to get by.**

**\* For Java 1.1, we'd use getMinimumSize().**

**\*/**

**public Dimension getMinimumSize() { return getPreferredSize(); }**

**/\*\***

**\* This method draws the component.**

**\* Note that it handles the margins and the alignment, but that**

**\* it doesn't have to worry about the color or font--the superclass**

**\* takes care of setting those in the Graphics object we're passed.**

**\*\*/**

**public void paint(Graphics g) {**

**int x, y;**

**Dimension size = this.getSize(); // use getSize() in Java 1.1**

**if (!measured) measure();**

**y = line\_ascent + (size.height - num\_lines \* line\_height)/2;**

**for(int i = 0; i < num\_lines; i++, y += line\_height) {**

**if (alignment == Alignment.LEFT) x = margin\_width;**

**else if (alignment == Alignment.CENTER)**

**x = (size.width - line\_widths[i])/2;**

**else x = size.width - margin\_width - line\_widths[i];**

**g.drawString(lines[i], x, y);**

**}**

**}**

**/\*\***

**\* This internal method breaks a specified label up into an array of lines.**

**\* It uses the StringTokenizer utility class.**

**\*\*/**

**protected synchronized void newLabel() {**

**StringTokenizer t = new StringTokenizer(label, "\n");**

**num\_lines = t.countTokens();**

**lines = new String[num\_lines];**

**line\_widths = new int[num\_lines];**

**for(int i = 0; i < num\_lines; i++) lines[i] = t.nextToken();**

**}**

**/\*\***

**\* This internal method figures out how the font is, and how wide each**

**\* line of the label is, and how wide the widest line is.**

**\*\*/**

**protected synchronized void measure() {**

**FontMetrics fm = this.getFontMetrics(this.getFont());**

**line\_height = fm.getHeight();**

**line\_ascent = fm.getAscent();**

**max\_width = 0;**

**for(int i = 0; i < num\_lines; i++) {**

**line\_widths[i] = fm.stringWidth(lines[i]);**

**if (line\_widths[i] > max\_width) max\_width = line\_widths[i];**

**}**

**measured = true;**

**}**

**}**

**A.8 Исходные коды лабораторной работы №8**

**Alignment.java**

**/\***

**\* To change this license header, choose License Headers in Project Properties.**

**\* To change this template file, choose Tools | Templates**

**\* and open the template in the editor.**

**\*/**

**package bsu.fpmi.educational\_practice2015;**

**/\*\***

**\***

**\* @author PC**

**\*/**

**public class Alignment {**

**private Alignment() {};**

**public static final Alignment LEFT = new Alignment();**

**public static final Alignment CENTER = new Alignment();**

**public static final Alignment RIGHT = new Alignment();**

**}**

**BasicPanel.java**

**/\***

**\* To change this license header, choose License Headers in Project Properties.**

**\* To change this template file, choose Tools | Templates**

**\* and open the template in the editor.**

**\*/**

**package bsu.fpmi.educational\_practice2015;**

**/\*\***

**\***

**\* @author PC**

**\*/**

**import java.awt.BorderLayout;**

**import java.awt.Button;**

**import java.awt.Dimension;**

**import java.awt.FlowLayout;**

**import java.awt.Panel;**

**import java.awt.event.ActionEvent;**

**import java.awt.event.ActionListener;**

**import java.awt.event.KeyEvent;**

**import java.awt.event.KeyListener;**

**import java.util.Vector;**

**import javax.swing.JCheckBox;**

**import javax.swing.JFrame;**

**import javax.swing.JLabel;**

**import javax.swing.JOptionPane;**

**import javax.swing.JTextField;**

**import javax.swing.WindowConstants;**

**public class BasicPanel extends Panel {**

**protected String messageText;**

**protected Alignment alignment;**

**protected String buttonLabel;**

**public char symbolOk = ':';**

**final static String messageS="Здесь находится статический текст";**

**protected JLabel message;**

**protected Button button;**

**protected JTextField textfield;**

**protected JCheckBox jcheckbox;**

**public BasicPanel() {**

**this("Symbol of confirmation is ");**

**}**

**public BasicPanel(String messageText) {**

**this(messageText, Alignment.LEFT, "Кнопка","","");**

**}**

**public BasicPanel(String messageText, Alignment alignment,**

**String buttonLabel, String r1ButtonLabel, String r2ButtonLabel)**

**{**

**setLayout(new BorderLayout(15, 15));**

**message = new JLabel();**

**add(message, BorderLayout.CENTER);**

**Panel buttonbox = new Panel();**

**buttonbox.setLayout(new FlowLayout(FlowLayout.CENTER, 25, 50));**

**add(buttonbox, BorderLayout.SOUTH);**

**textfield =new JTextField();**

**textfield.setPreferredSize(new Dimension(200,24));**

**buttonbox.add(textfield);**

**jcheckbox=new JCheckBox("Кнопка-флаг");**

**buttonbox.add(jcheckbox);**

**button = new Button();**

**buttonbox.add(button);**

**button.addActionListener(new ActionListener() {**

**public void actionPerformed(ActionEvent e) {**

**fireEvent(new ButtonEvent(BasicPanel.this,**

**ButtonEvent.button, jcheckbox.isSelected(), textfield.getText()));**

**}**

**});**

**setMessageText(messageText);**

**setButtonLabel(buttonLabel);**

**setSymbolOk(symbolOk);**

**addButtonListener(new ButtonListener() {**

**public void button(ButtonEvent e) {**

**JOptionPane.showMessageDialog(null,"Введенный текст: " + e.text+ "\n"+"Состояние флага: "+ e.flag );**

**button.setFocusable(false);**

**message.setFocusable(true);**

**}**

**});**

**message.setFocusable(true);**

**message.addKeyListener(new KeyListener(){**

**public void keyTyped(KeyEvent e) {**

**}**

**@Override**

**public void keyPressed(KeyEvent e) {**

**if(Character.compare(e.getKeyChar(), symbolOk)==0 || message.hasFocus()|| button.hasFocus())**

**{**

**JOptionPane.showMessageDialog(null,"Введенный текст: " + textfield.getText()+ "\n"+"Состояние флага: "+ jcheckbox.isSelected() );**

**}**

**message.setFocusable(true);**

**}**

**@Override**

**public void keyReleased(KeyEvent e) {**

**}**

**});**

**}**

**public String getMessageText() { return messageText; }**

**public String getButtonLabel() { return buttonLabel; }**

**public char getSymbolOk(){return symbolOk;}**

**public void setMessageText(String messageText) {**

**this.messageText = messageText;**

**message.setText(messageText);**

**validate();**

**}**

**public void setButtonLabel(String l) {**

**buttonLabel = l;**

**button.setLabel(l);**

**button.setVisible((l != null) && (l.length() > 0));**

**validate();**

**}**

**@SuppressWarnings("deprecation")**

**public void setSymbolOk(char l){**

**this.symbolOk=l;**

**setMessageText(messageS+Character.toString(l));**

**validate();**

**}**

**protected Vector<ButtonListener> listeners = new Vector<ButtonListener>();**

**public void addButtonListener(ButtonListener l) {**

**listeners.addElement(l);**

**}**

**public void removeButtonListener(ButtonListener l) {**

**listeners.removeElement(l);**

**}**

**public void fireEvent(ButtonEvent e) {**

**Vector list = (Vector) listeners.clone();**

**for(int i = 0; i < list.size(); i++) {**

**ButtonListener listener = (ButtonListener)list.elementAt(i);**

**switch(e.getID()) {**

**case ButtonEvent.button: listener.button(e); break;**

**}**

**}**

**}**

**public static void main(String[] args) {**

**BasicPanel p = new BasicPanel();**

**p.setSymbolOk(':');**

**JFrame f = new JFrame();**

**f.add(p);**

**f.pack();**

**f.setVisible(true);**

**f.setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);**

**}**

**}**

**BasicPanelBeanInfo.java**

**/\***

**\* To change this license header, choose License Headers in Project Properties.**

**\* To change this template file, choose Tools | Templates**

**\* and open the template in the editor.**

**\*/**

**package bsu.fpmi.educational\_practice2015;**

**/\*\***

**\***

**\* @author PC**

**\*/**

**import java.beans.\*;**

**import java.lang.reflect.\*;**

**import java.awt.\*;**

**public class BasicPanelBeanInfo extends SimpleBeanInfo {**

**public Image getIcon(int kind) { return loadImage("Image.gif"); }**

**static PropertyDescriptor prop(String name, String description) {**

**try {**

**PropertyDescriptor p =**

**new PropertyDescriptor(name, BasicPanel.class);**

**p.setShortDescription(description);**

**return p;**

**}**

**catch(IntrospectionException e) { return null; }**

**}**

**static PropertyDescriptor[] props = {**

**prop("messageText", "The message text that appears in the bean body"),**

**prop("buttonLabel", "The label for the basic button"),**

**prop("symbolOk", "The symbol of confirmation for actions"),**

**};**

**public PropertyDescriptor[] getPropertyDescriptors() { return props; }**

**public int getDefaultPropertyIndex() { return 0; }**

**}**

**ButtonEvent.java**

**/\***

**\* To change this license header, choose License Headers in Project Properties.**

**\* To change this template file, choose Tools | Templates**

**\* and open the template in the editor.**

**\*/**

**package bsu.fpmi.educational\_practice2015;**

**/\*\***

**\***

**\* @author PC**

**\*/**

**public class ButtonEvent extends java.util.EventObject {**

**public static final int button = 0, symbol=1;**

**protected int id;**

**boolean flag;**

**String text;**

**public ButtonEvent(Object source, int id, boolean f, String t) {**

**super(source);**

**this.id = id;**

**flag=f;**

**text=t;**

**}**

**public int getID() {**

**return id;**

**}**

**}**

**ButtonListener.java**

**/\***

**\* To change this license header, choose License Headers in Project Properties.**

**\* To change this template file, choose Tools | Templates**

**\* and open the template in the editor.**

**\*/**

**package bsu.fpmi.educational\_practice2015;**

**/\*\***

**\***

**\* @author PC**

**\*/**

**public interface ButtonListener extends java.util.EventListener {**

**public void button(ButtonEvent e);**

**}**

**NewJPanel.java**

**/\***

**\* To change this license header, choose License Headers in Project Properties.**

**\* To change this template file, choose Tools | Templates**

**\* and open the template in the editor.**

**\*/**

**package bsu.fpmi.educational\_practice2015;**

**import java.awt.Button;**

**/\*\***

**\***

**\* @author fpm.grabovoy**

**\*/**

**public class NewJPanel extends javax.swing.JPanel {**

**/\*\***

**\* Creates new form NewJPanel**

**\*/**

**public NewJPanel() {**

**initComponents();**

**}**

**/\*\***

**\* This method is called from within the constructor to initialize the form.**

**\* WARNING: Do NOT modify this code. The content of this method is always**

**\* regenerated by the Form Editor.**

**\*/**

**@SuppressWarnings("unchecked")**

**// <editor-fold defaultstate="collapsed" desc="Generated Code">**

**private void initComponents() {**

**basicPanel1 = new bsu.fpmi.educational\_practice2015.BasicPanel();**

**jButton1 = new javax.swing.JButton();**

**basicPanel2 = new bsu.fpmi.educational\_practice2015.BasicPanel();**

**jButton1.setText("jButton1");**

**jButton1.addActionListener(new java.awt.event.ActionListener() {**

**public void actionPerformed(java.awt.event.ActionEvent evt) {**

**jButton1ActionPerformed(evt);**

**}**

**});**

**basicPanel2.setButtonLabel("999");**

**basicPanel2.setMessageText("999");**

**javax.swing.GroupLayout layout = new javax.swing.GroupLayout(this);**

**this.setLayout(layout);**

**layout.setHorizontalGroup(**

**layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)**

**.addGroup(layout.createSequentialGroup()**

**.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)**

**.addGroup(layout.createSequentialGroup()**

**.addGap(940, 940, 940)**

**.addComponent(jButton1))**

**.addComponent(basicPanel2, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))**

**.addGap(0, 0, Short.MAX\_VALUE))**

**);**

**layout.setVerticalGroup(**

**layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)**

**.addGroup(layout.createSequentialGroup()**

**.addContainerGap()**

**.addComponent(basicPanel2, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)**

**.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)**

**.addComponent(jButton1)**

**.addContainerGap(215, Short.MAX\_VALUE))**

**);**

**}// </editor-fold>**

**private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {**

**// TODO add your handling code here:**

**}**

**// Variables declaration - do not modify**

**private bsu.fpmi.educational\_practice2015.BasicPanel basicPanel1;**

**private bsu.fpmi.educational\_practice2015.BasicPanel basicPanel2;**

**private javax.swing.JButton jButton1;**

**// End of variables declaration**

**}**

**A.9 Исходные коды лабораторной работы №9**

**BasicPanel.java**

**package bsu.fpmi.educational\_practice2015;**

**import java.awt.BorderLayout;**

**import java.awt.Button;**

**import java.awt.Dimension;**

**import java.awt.FlowLayout;**

**import java.awt.Panel;**

**import java.awt.event.ActionEvent;**

**import java.awt.event.ActionListener;**

**import java.awt.event.KeyEvent;**

**import java.awt.event.KeyListener;**

**import java.util.Vector;**

**import javax.swing.JCheckBox;**

**import javax.swing.JFrame;**

**import javax.swing.JLabel;**

**import javax.swing.JOptionPane;**

**import javax.swing.JTextField;**

**import javax.swing.WindowConstants;**

**public class BasicPanel extends Panel {**

**protected String labelText = "Здесь находится статический текст";**

**protected String buttonLabel= "Кнопка";**

**protected String jcheckboxLabel= "Кнопка флаг";**

**final static String lbT="Здесь находится статический текст";**

**public char symbolOk = ':';**

**protected JLabel label = new JLabel(labelText);**

**protected Button button = new Button(buttonLabel);**

**protected JCheckBox jcheckbox = new JCheckBox(jcheckboxLabel);**

**protected JTextField textfield;**

**public BasicPanel() {**

**setLayout(new BorderLayout(15, 15));**

**this.setBackground(java.awt.Color.lightGray);**

**add(label, BorderLayout.CENTER);**

**Panel buttonbox = new Panel();**

**buttonbox.setLayout(new FlowLayout(FlowLayout.CENTER, 25, 15));**

**add(buttonbox, BorderLayout.SOUTH);**

**textfield =new JTextField();**

**textfield.setPreferredSize(new Dimension(200,24));**

**buttonbox.add(textfield);**

**buttonbox.add(jcheckbox);**

**buttonbox.add(button);**

**button.addActionListener(new ActionListener() {**

**public void actionPerformed(ActionEvent e) {**

**fireEvent(new ButtonEvent(BasicPanel.this,**

**ButtonEvent.button));**

**}**

**});**

**setSymbolOk(symbolOk);**

**addButtonListener(new ButtonListener() {**

**public void button(ButtonEvent e) {**

**JOptionPane.showMessageDialog(null,"Введенный текст: " + textfield.getText()+ "\n"+"Состояние флага: "+ jcheckbox.isSelected() );**

**button.setFocusable(false);**

**label.setFocusable(true);**

**}**

**});**

**label.setFocusable(true);**

**label.addKeyListener(new KeyListener(){**

**public void keyTyped(KeyEvent e) {**

**}**

**@Override**

**public void keyPressed(KeyEvent e) {**

**if(Character.compare(e.getKeyChar(), symbolOk)==0 || label.hasFocus()|| button.hasFocus())**

**{**

**JOptionPane.showMessageDialog(null,"Введенный текст: " + textfield.getText()+ "\n"+"Состояние флага: "+ jcheckbox.isSelected() );**

**}**

**label.setFocusable(true);**

**}**

**@Override**

**public void keyReleased(KeyEvent e) {**

**}**

**});**

**buttonbox.setVisible(true);**

**}**

**public String getLabelText() { return labelText; }**

**public String getButtonLabel() { return buttonLabel; }**

**//public String getJCheckBoxLabel() { return jcheckboxLabel;}**

**public void setLabelText(String messageText) {**

**this.labelText = messageText;**

**label.setText(messageText);**

**validate();**

**}**

**// public void setJCheckBoxLabel(String Text) {**

**// this.jcheckboxLabel = Text;**

**// jcheckbox.setLabel(Text);**

**// validate();**

**// }**

**public void setButtonLabel(String l) {**

**buttonLabel = l;**

**button.setLabel(l);**

**button.setVisible((l != null) && (l.length() > 0));**

**validate();**

**}**

**@SuppressWarnings("deprecation")**

**public void setSymbolOk(char l){**

**this.symbolOk=l;**

**setLabelText(lbT+Character.toString(l));**

**//validate();**

**}**

**protected Vector<ButtonListener> listeners = new Vector<ButtonListener>();**

**public void addButtonListener(ButtonListener l) {**

**listeners.addElement(l);**

**}**

**public void removeButtonListener(ButtonListener l) {**

**listeners.removeElement(l);**

**}**

**public void fireEvent(ButtonEvent e) {**

**Vector list = (Vector) listeners.clone();**

**for(int i = 0; i < list.size(); i++) {**

**ButtonListener listener = (ButtonListener)list.elementAt(i);**

**switch(e.getID()) {**

**case ButtonEvent.button: listener.button(e); break;**

**}**

**}**

**}**

**public static void main(String[] args) {**

**BasicPanel p = new BasicPanel();**

**p.setSymbolOk(':');**

**p.setVisible(true);**

**JFrame f = new JFrame();**

**f.add(p);**

**f.pack();**

**f.setVisible(true);**

**f.setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);**

**}**

**}**

**BasicPanelBeanInfo.java**

**package bsu.fpmi.educational\_practice2015;**

**import java.beans.BeanDescriptor;**

**import java.beans.IntrospectionException;**

**import java.beans.PropertyDescriptor;**

**import java.beans.SimpleBeanInfo;**

**public class BasicPanelBeanInfo extends SimpleBeanInfo{**

**@Override**

**public BeanDescriptor getBeanDescriptor() {**

**return new BeanDescriptor(BasicPanel.class,**

**BasicPanelCustomizer.class);**

**}**

**@Override**

**public PropertyDescriptor[] getPropertyDescriptors() {**

**return properties;**

**}**

**@Override**

**public int getDefaultPropertyIndex() {**

**return 0;**

**}**

**private static final PropertyDescriptor[] properties = {**

**property("labelText", "Text of the label", LabelTextEditor.class),**

**property("buttonLabel", "Text of the button", ButtonTextEditor.class),**

**//property("jcheckboxLabel", "Text of the flag", JCheckBoxTextEditor.class),**

**};**

**private static PropertyDescriptor property(String name, String description, Class<?> propertyEditorClass) {**

**try {**

**PropertyDescriptor descriptor =**

**new PropertyDescriptor(name, BasicPanel.class);**

**descriptor.setShortDescription(description);**

**if (propertyEditorClass != null) {**

**// JOptionPane.showMessageDialog(null, propertyEditorClass + "\nAAAAAAAAAAAA!!!!");**

**descriptor.setPropertyEditorClass(propertyEditorClass);**

**}**

**return descriptor;**

**} catch (IntrospectionException e) {**

**return null;**

**}**

**}**

**}**

**BasicPanelCustomizer.java**

**package bsu.fpmi.educational\_practice2015;**

**import java.awt.\*;**

**import java.awt.event.\*;**

**import java.beans.\*;**

**public class BasicPanelCustomizer extends Panel**

**implements Customizer, TextListener**

**{**

**protected BasicPanel bean;**

**protected TextArea message;**

**protected TextField fields[];**

**public void setObject(Object o) {**

**bean = (BasicPanel)o;**

**this.setLayout(new BorderLayout());**

**this.add(new Label("Enter the of the label:"), "North");**

**message = new TextArea(bean.getLabelText());**

**message.addTextListener(this);**

**message.setSize(400, 200);**

**this.add(message, "Center");**

**Panel buttonbox = new Panel();**

**buttonbox.setLayout(new GridLayout(1, 0, 25, 10));**

**this.add(buttonbox, "South");**

**fields = new TextField[2];**

**String[] labels = new String[] {"Button text"};**

**String[] values = new String[] { bean.getButtonLabel()};**

**// String[] labels1 = new String[] {"JCheckBox text"};**

**// String[] values1 = new String[] { bean.getJCheckBoxLabel()};**

**Panel p = new Panel();**

**p.setLayout(new BorderLayout());**

**p.add(new Label(labels[0]), "North");**

**fields[1] = new TextField(values[1]);**

**p.add(fields[1], "Center");**

**fields[1].addTextListener(this);**

**// p.add(new Label(labels1[0]), "North");**

**// fields[2] = new TextField(values1[1]);**

**// p.add(fields[2], "Center");**

**// fields[2].addTextListener(this);**

**buttonbox.add(p);**

**}**

**public Insets getInsets() { return new Insets(10, 10, 10, 10); }**

**public void textValueChanged(TextEvent e) {**

**TextComponent t = (TextComponent)e.getSource();**

**String s = t.getText();**

**if (t == message) bean.setLabelText(s);**

**else if (t == fields[1]) bean.setButtonLabel(s);**

**//else if (t == fields[2]) bean.setJCheckBoxLabel(s);**

**listeners.firePropertyChange(null, null, null);**

**}**

**protected PropertyChangeSupport listeners = new PropertyChangeSupport(this);**

**public void addPropertyChangeListener(PropertyChangeListener l) {**

**listeners.addPropertyChangeListener(l);**

**}**

**public void removePropertyChangeListener(PropertyChangeListener l) {**

**listeners.removePropertyChangeListener(l);**

**}**

**}**

**ButtonEvent.java**

**package bsu.fpmi.educational\_practice2015;**

**public class ButtonEvent extends java.util.EventObject {**

**public static final int button = 0, symbol=1;**

**protected int id;**

**public ButtonEvent(Object source, int id) {**

**super(source);**

**this.id = id;**

**}**

**public int getID() {**

**return id;**

**}**

**}**

**ButtonListener.java**

**package bsu.fpmi.educational\_practice2015;**

**public interface ButtonListener extends java.util.EventListener {**

**public void button(ButtonEvent e);**

**}**

**ButtonTextEditor.java**

**package bsu.fpmi.educational\_practice2015;**

**import java.beans.PropertyEditorSupport;**

**public class ButtonTextEditor extends PropertyEditorSupport{**

**@Override**

**public String[] getTags() {**

**return new String[] {"Вы уверены?", "Жмякай", "Кнопка","Пуск"};**

**}**

**@Override**

**public String getJavaInitializationString() {**

**return "\"" + ((String)getValue()).replace("\"", "\\\"") + "\"";**

**}**

**}**

**JCheckBoxTextEditor.java**

**package bsu.fpmi.educational\_practice2015;**

**import java.beans.PropertyEditorSupport;**

**public class ButtonTextEditor extends PropertyEditorSupport{**

**@Override**

**public String[] getTags() {**

**return new String[] {"Вы уверены?", "Жмякай", "Кнопка","Пуск"};**

**}**

**@Override**

**public String getJavaInitializationString() {**

**return "\"" + ((String)getValue()).replace("\"", "\\\"") + "\"";**

**}**

**}**

**LabelTextEditor.java**

**package bsu.fpmi.educational\_practice2015;**

**import java.beans.PropertyEditorSupport;**

**public class ButtonTextEditor extends PropertyEditorSupport{**

**@Override**

**public String[] getTags() {**

**return new String[] {"Вы уверены?", "Жмякай", "Кнопка","Пуск"};**

**}**

**@Override**

**public String getJavaInitializationString() {**

**return "\"" + ((String)getValue()).replace("\"", "\\\"") + "\"";**

**}**

**}**

**A.10 Исходные коды лабораторной работы №10**

**MainJFrame.java**

**/\***

**\* To change this license header, choose License Headers in Project Properties.**

**\* To change this template file, choose Tools | Templates**

**\* and open the template in the editor.**

**\*/**

**package Frame;**

**import java.awt.event.WindowAdapter;**

**import java.awt.event.WindowEvent;**

**import java.io.File;**

**import java.util.Scanner;**

**public class MainJFrame extends javax.swing.JFrame {**

**public MainJFrame() {**

**initComponents();**

**addWindowListener(new WindowAdapter() {**

**public void windowClosing(WindowEvent e)**

**{**

**System.exit(0);**

**}**

**});**

**}**

**@SuppressWarnings("unchecked")**

**// <editor-fold defaultstate="collapsed" desc="Generated Code">**

**private void initComponents() {**

**jScrollPane1 = new javax.swing.JScrollPane();**

**jTextArea1 = new javax.swing.JTextArea();**

**jPanel1 = new javax.swing.JPanel();**

**jFileChooser1 = new javax.swing.JFileChooser();**

**jMenuBar1 = new javax.swing.JMenuBar();**

**jMenu1 = new javax.swing.JMenu();**

**jMenuItem1 = new javax.swing.JMenuItem();**

**jMenuItem2 = new javax.swing.JMenuItem();**

**jMenu2 = new javax.swing.JMenu();**

**jMenuItem3 = new javax.swing.JMenuItem();**

**jMenuItem4 = new javax.swing.JMenuItem();**

**jMenu3 = new javax.swing.JMenu();**

**jMenuItem5 = new javax.swing.JMenuItem();**

**setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);**

**jTextArea1.setColumns(20);**

**jTextArea1.setRows(5);**

**jScrollPane1.setViewportView(jTextArea1);**

**jFileChooser1.setCurrentDirectory(new java.io.File("C:\\Users\\Валентина\\Desktop\\up-10"));**

**jFileChooser1.setDialogTitle("Choose File");**

**jFileChooser1.setFileFilter(null);**

**javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);**

**jPanel1.setLayout(jPanel1Layout);**

**jPanel1Layout.setHorizontalGroup(**

**jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)**

**.addGroup(jPanel1Layout.createSequentialGroup()**

**.addGap(28, 28, 28)**

**.addComponent(jFileChooser1, javax.swing.GroupLayout.PREFERRED\_SIZE, 566, javax.swing.GroupLayout.PREFERRED\_SIZE)**

**.addContainerGap(42, Short.MAX\_VALUE))**

**);**

**jPanel1Layout.setVerticalGroup(**

**jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)**

**.addGroup(javax.swing.GroupLayout.Alignment.TRAILING, jPanel1Layout.createSequentialGroup()**

**.addContainerGap(175, Short.MAX\_VALUE)**

**.addComponent(jFileChooser1, javax.swing.GroupLayout.PREFERRED\_SIZE, 101, javax.swing.GroupLayout.PREFERRED\_SIZE)**

**.addContainerGap())**

**);**

**jMenu1.setText("File");**

**jMenuItem1.setText("Open");**

**jMenuItem1.addActionListener(new java.awt.event.ActionListener() {**

**public void actionPerformed(java.awt.event.ActionEvent evt) {**

**jMenuItem1ActionPerformed(evt);**

**}**

**});**

**jMenu1.add(jMenuItem1);**

**jMenuItem2.setText("Quit");**

**jMenu1.add(jMenuItem2);**

**jMenuBar1.add(jMenu1);**

**jMenu2.setText("View");**

**jMenuItem3.setText("Font");**

**jMenu2.add(jMenuItem3);**

**jMenuItem4.setText("Look and Feel");**

**jMenu2.add(jMenuItem4);**

**jMenuBar1.add(jMenu2);**

**jMenu3.setText("Help");**

**jMenuItem5.setText("About");**

**jMenuItem5.addActionListener(new java.awt.event.ActionListener() {**

**public void actionPerformed(java.awt.event.ActionEvent evt) {**

**jMenuItem5ActionPerformed(evt);**

**}**

**});**

**jMenu3.add(jMenuItem5);**

**jMenuBar1.add(jMenu3);**

**setJMenuBar(jMenuBar1);**

**javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());**

**getContentPane().setLayout(layout);**

**layout.setHorizontalGroup(**

**layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)**

**.addComponent(jScrollPane1)**

**.addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)**

**);**

**layout.setVerticalGroup(**

**layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)**

**.addGroup(layout.createSequentialGroup()**

**.addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED\_SIZE, 172, javax.swing.GroupLayout.PREFERRED\_SIZE)**

**.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)**

**.addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))**

**);**

**pack();**

**}// </editor-fold>**

**protected String Text;**

**private void jMenuItem1ActionPerformed(java.awt.event.ActionEvent evt) {**

**try**

**{**

**this.jFileChooser1.show();**

**int res = this.jFileChooser1.showOpenDialog(null);**

**if(res == 0)**

**{**

**File f = this.jFileChooser1.getSelectedFile();**

**Scanner fin = new Scanner( f );**

**while(fin.hasNextLine())**

**{**

**String str = fin.nextLine();**

**Text+=str;**

**}**

**this.jTextArea1.setText(Text);**

**}**

**else { }**

**}**

**catch(Exception e) { }**

**}**

**private void jMenuItem5ActionPerformed(java.awt.event.ActionEvent evt) {**

**// TODO add your handling code here:**

**}**

**public static void main(String args[]) {**

**/\* Set the Nimbus look and feel \*/**

**//<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">**

**/\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.**

**\* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html**

**\*/**

**try {**

**for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {**

**if ("Nimbus".equals(info.getName())) {**

**javax.swing.UIManager.setLookAndFeel(info.getClassName());**

**break;**

**}**

**}**

**} catch (ClassNotFoundException ex) {**

**java.util.logging.Logger.getLogger(MainJFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);**

**} catch (InstantiationException ex) {**

**java.util.logging.Logger.getLogger(MainJFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);**

**} catch (IllegalAccessException ex) {**

**java.util.logging.Logger.getLogger(MainJFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);**

**} catch (javax.swing.UnsupportedLookAndFeelException ex) {**

**java.util.logging.Logger.getLogger(MainJFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);**

**}**

**//</editor-fold>**

**/\* Create and display the form \*/**

**java.awt.EventQueue.invokeLater(new Runnable() {**

**public void run() {**

**new MainJFrame().setVisible(true);**

**}**

**});**

**}**

**// Variables declaration - do not modify**

**private javax.swing.JFileChooser jFileChooser1;**

**private javax.swing.JMenu jMenu1;**

**private javax.swing.JMenu jMenu2;**

**private javax.swing.JMenu jMenu3;**

**private javax.swing.JMenuBar jMenuBar1;**

**private javax.swing.JMenuItem jMenuItem1;**

**private javax.swing.JMenuItem jMenuItem2;**

**private javax.swing.JMenuItem jMenuItem3;**

**private javax.swing.JMenuItem jMenuItem4;**

**private javax.swing.JMenuItem jMenuItem5;**

**private javax.swing.JPanel jPanel1;**

**private javax.swing.JScrollPane jScrollPane1;**

**private javax.swing.JTextArea jTextArea1;**

**// End of variables declaration**

**}**

**FontChooser.java**

**package SelfMade;**

**import java.awt.BorderLayout;**

**import java.awt.Container;**

**import java.awt.Font;**

**import java.awt.Frame;**

**import java.awt.GraphicsEnvironment;**

**import java.awt.event.ActionEvent;**

**import java.awt.event.ActionListener;**

**import javax.swing.Box;**

**import javax.swing.JButton;**

**import javax.swing.JDialog;**

**import javax.swing.JScrollPane;**

**import javax.swing.JTextArea;**

**public class FontChooser extends JDialog**

**{**

**// These fields define the component properties**

**String family; // The name of the font family**

**int style; // The font style**

**int size; // The font size**

**Font selectedFont; // The Font they correspond to**

**// This is the list of all font families on the system**

**String[] fontFamilies;**

**// The various Swing components used in the dialog**

**ItemChooser families, styles, sizes;**

**JTextArea preview;**

**JButton okay, cancel;**

**// The names to appear in the "Style" menu**

**static final String[] styleNames = new String[] {**

**"Plain", "Italic", "Bold", "BoldItalic"**

**};**

**// The style values that correspond to those names**

**static final Integer[] styleValues = new Integer[] {**

**new Integer(Font.PLAIN), new Integer(Font.ITALIC),**

**new Integer(Font.BOLD), new Integer(Font.BOLD+Font.ITALIC)**

**};**

**// The size "names" to appear in the size menu**

**static final String[] sizeNames = new String[] {**

**"8", "10", "12", "14", "18", "20", "24", "28", "32",**

**"40", "48", "56", "64", "72"**

**};**

**// This is the default preview string displayed in the dialog box**

**static final String defaultPreviewString =**

**"ABCDEFGHIJKLMNOPQRSTUVWXYZ\n" +**

**"abcdefghijklmnopqrstuvwxyz\n" +**

**"1234567890!@#$%^&\*()\_-=+[]{}<,.>\n" +**

**"The quick brown fox jumps over the lazy dog";**

**/\*\* Create a font chooser dialog for the specified frame. \*/**

**public FontChooser(Frame owner) {**

**super(owner, "Choose a Font"); // Set dialog frame and title**

**// This dialog must be used as a modal dialog. In order to be used**

**// as a modeless dialog, it would have to fire a PropertyChangeEvent**

**// whenever the selected font changed, so that applications could be**

**// notified of the user's selections.**

**setModal(true);**

**// Figure out what fonts are available on the system**

**GraphicsEnvironment env =**

**GraphicsEnvironment.getLocalGraphicsEnvironment();**

**fontFamilies = env.getAvailableFontFamilyNames();**

**// Set initial values for the properties**

**family = fontFamilies[0];**

**style = Font.PLAIN;**

**size = 18;**

**selectedFont = new Font(family, style, size);**

**// Create ItemChooser objects that allow the user to select font**

**// family, style, and size.**

**families = new ItemChooser("Family", fontFamilies, null, 0,**

**ItemChooser.COMBOBOX);**

**styles = new ItemChooser("Style", styleNames, styleValues, 0,**

**ItemChooser.COMBOBOX);**

**sizes = new ItemChooser("Size", sizeNames,null,4,ItemChooser.COMBOBOX);**

**// Now register event listeners to handle selections**

**families.addItemChooserListener(new ItemChooser.Listener() {**

**public void itemChosen(ItemChooser.Event e) {**

**setFontFamily((String)e.getSelectedValue());**

**}**

**});**

**styles.addItemChooserListener(new ItemChooser.Listener() {**

**public void itemChosen(ItemChooser.Event e) {**

**setFontStyle(((Integer)e.getSelectedValue()).intValue());**

**}**

**});**

**sizes.addItemChooserListener(new ItemChooser.Listener() {**

**public void itemChosen(ItemChooser.Event e) {**

**setFontSize(Integer.parseInt((String)e.getSelectedValue()));**

**}**

**});**

**// Create a component to preview the font.**

**preview = new JTextArea(defaultPreviewString, 5, 40);**

**preview.setFont(selectedFont);**

**// Create buttons to dismiss the dialog, and set handlers on them**

**okay = new JButton("Okay");**

**cancel = new JButton("Cancel");**

**okay.addActionListener(new ActionListener() {**

**public void actionPerformed(ActionEvent e) { setVisible(false); }**

**});**

**cancel.addActionListener(new ActionListener() {**

**public void actionPerformed(ActionEvent e) {**

**selectedFont = null;**

**setVisible(false);**

**}**

**});**

**// Put the ItemChoosers in a Box**

**Box choosersBox = Box.createHorizontalBox();**

**choosersBox.add(Box.createHorizontalStrut(15));**

**choosersBox.add(families);**

**choosersBox.add(Box.createHorizontalStrut(15));**

**choosersBox.add(styles);**

**choosersBox.add(Box.createHorizontalStrut(15));**

**choosersBox.add(sizes);**

**choosersBox.add(Box.createHorizontalStrut(15));**

**choosersBox.add(Box.createGlue());**

**// Put the dismiss buttons in another box**

**Box buttonBox = Box.createHorizontalBox();**

**buttonBox.add(Box.createGlue());**

**buttonBox.add(okay);**

**buttonBox.add(Box.createGlue());**

**buttonBox.add(cancel);**

**buttonBox.add(Box.createGlue());**

**// Put the choosers at the top, the buttons at the bottom, and**

**// the preview in the middle.**

**Container contentPane = getContentPane();**

**contentPane.add(new JScrollPane(preview), BorderLayout.CENTER);**

**contentPane.add(choosersBox, BorderLayout.NORTH);**

**contentPane.add(buttonBox, BorderLayout.SOUTH);**

**// Set the dialog size based on the component size.**

**pack();**

**}**

**/\*\***

**\* Call this method after show() to obtain the user's selection. If the**

**\* user used the "Cancel" button, this will return null**

**\*\*/**

**public Font getSelectedFont() { return selectedFont; }**

**// These are other property getter methods**

**public String getFontFamily() { return family; }**

**public int getFontStyle() { return style; }**

**public int getFontSize() { return size; }**

**// The property setter methods are a little more complicated.**

**// Note that none of these setter methods update the corresponding**

**// ItemChooser components as they ought to.**

**public void setFontFamily(String name) {**

**family = name;**

**changeFont();**

**}**

**public void setFontStyle(int style) {**

**this.style = style;**

**changeFont();**

**}**

**public void setFontSize(int size) {**

**this.size = size;**

**changeFont();**

**}**

**public void setSelectedFont(Font font) {**

**selectedFont = font;**

**family = font.getFamily();**

**style = font.getStyle();**

**size = font.getSize();**

**preview.setFont(font);**

**}**

**// This method is called when the family, style, or size changes**

**protected void changeFont() {**

**selectedFont = new Font(family, style, size);**

**preview.setFont(selectedFont);**

**}**

**// Override this inherited method to prevent anyone from making us modeless**

**public boolean isModal() { return true; }**

**}**

**ItemChooser.java**

**package SelfMade;**

**import java.awt.\*;**

**import java.awt.event.\*;**

**import javax.swing.\*;**

**import javax.swing.event.\*;**

**import javax.swing.border.\*;**

**import java.util.\*;**

**public class ItemChooser extends JPanel**

**{**

**String name; // The overall name of the choice**

**String[] labels; // The text for each choice option**

**Object[] values; // Arbitrary values associated with each option**

**int selection; // The selected choice**

**int presentation; // How the choice is presented**

**// These are the legal values for the presentation field**

**public static final int LIST = 1;**

**public static final int COMBOBOX = 2;**

**public static final int RADIOBUTTONS = 3;**

**// These components are used for each of the 3 possible presentations**

**JList<String> list; // One type of presentation**

**JComboBox<String> combobox; // Another type of presentation**

**JRadioButton[] radiobuttons; // Yet another type**

**// The list of objects that are interested in our state**

**ArrayList<ItemChooser.Listener> listeners = new ArrayList<ItemChooser.Listener>();**

**// The constructor method sets everything up**

**public ItemChooser(String name, String[] labels, Object[] values,**

**int defaultSelection, int presentation)**

**{**

**// Copy the constructor arguments to instance fields**

**this.name = name;**

**this.labels = labels;**

**this.values = values;**

**this.selection = defaultSelection;**

**this.presentation = presentation;**

**// If no values were supplied, use the labels**

**if (values == null) this.values = labels;**

**// Now create content and event handlers based on presentation type**

**switch(presentation) {**

**case LIST: initList(); break;**

**case COMBOBOX: initComboBox(); break;**

**case RADIOBUTTONS: initRadioButtons(); break;**

**}**

**}**

**// Initialization for JList presentation**

**void initList() {**

**list = new JList<String>(labels); // Create the list**

**list.setSelectedIndex(selection); // Set initial state**

**// Handle state changes**

**list.addListSelectionListener(new ListSelectionListener() {**

**public void valueChanged(ListSelectionEvent e) {**

**ItemChooser.this.select(list.getSelectedIndex());**

**}**

**});**

**// Lay out list and name label vertically**

**this.setLayout(new BoxLayout(this, BoxLayout.Y\_AXIS)); // vertical**

**this.add(new JLabel(name)); // Display choice name**

**this.add(new JScrollPane(list)); // Add the JList**

**}**

**// Initialization for JComboBox presentation**

**void initComboBox() {**

**combobox = new JComboBox<String>(labels); // Create the combo box**

**combobox.setSelectedIndex(selection); // Set initial state**

**// Handle changes to the state**

**combobox.addItemListener(new ItemListener() {**

**public void itemStateChanged(ItemEvent e) {**

**ItemChooser.this.select(combobox.getSelectedIndex());**

**}**

**});**

**// Lay out combo box and name label horizontally**

**this.setLayout(new BoxLayout(this, BoxLayout.X\_AXIS));**

**this.add(new JLabel(name));**

**this.add(combobox);**

**}**

**// Initialization for JRadioButton presentation**

**void initRadioButtons() {**

**// Create an array of mutually exclusive radio buttons**

**radiobuttons = new JRadioButton[labels.length]; // the array**

**ButtonGroup radioButtonGroup = new ButtonGroup(); // used for exclusion**

**ChangeListener listener = new ChangeListener() { // A shared listener**

**public void stateChanged(ChangeEvent e) {**

**JRadioButton b = (JRadioButton)e.getSource();**

**if (b.isSelected()) {**

**// If we received this event because a button was**

**// selected, then loop through the list of buttons to**

**// figure out the index of the selected one.**

**for(int i = 0; i < radiobuttons.length; i++) {**

**if (radiobuttons[i] == b) {**

**ItemChooser.this.select(i);**

**return;**

**}**

**}**

**}**

**}**

**};**

**// Display the choice name in a border around the buttons**

**this.setBorder(new TitledBorder(new EtchedBorder(), name));**

**this.setLayout(new BoxLayout(this, BoxLayout.Y\_AXIS));**

**// Create the buttons, add them to the button group, and specify**

**// the event listener for each one.**

**for(int i = 0; i < labels.length; i++) {**

**radiobuttons[i] = new JRadioButton(labels[i]);**

**if (i == selection) radiobuttons[i].setSelected(true);**

**radiobuttons[i].addChangeListener(listener);**

**radioButtonGroup.add(radiobuttons[i]);**

**this.add(radiobuttons[i]);**

**}**

**}**

**// These simple property accessor methods just return field values**

**// These are read-only properties. The values are set by the constructor**

**// and may not be changed.**

**public String getName() { return name; }**

**public int getPresentation() { return presentation; }**

**public String[] getLabels() { return labels; }**

**public Object[] getValues() { return values; }**

**/\*\* Return the index of the selected item \*/**

**public int getSelectedIndex() { return selection; }**

**/\*\* Return the object associated with the selected item \*/**

**public Object getSelectedValue() { return values[selection]; }**

**/\*\***

**\* Set the selected item by specifying its index. Calling this**

**\* method changes the on-screen display but does not generate events.**

**\*\*/**

**public void setSelectedIndex(int selection) {**

**switch(presentation) {**

**case LIST: list.setSelectedIndex(selection); break;**

**case COMBOBOX: combobox.setSelectedIndex(selection); break;**

**case RADIOBUTTONS: radiobuttons[selection].setSelected(true); break;**

**}**

**this.selection = selection;**

**}**

**/\*\***

**\* This internal method is called when the selection changes. It stores**

**\* the new selected index, and fires events to any registered listeners.**

**\* The event listeners registered on the JList, JComboBox, or JRadioButtons**

**\* all call this method.**

**\*\*/**

**protected void select(int selection) {**

**this.selection = selection; // Store the new selected index**

**if (!listeners.isEmpty()) { // If there are any listeners registered**

**// Create an event object to describe the selection**

**ItemChooser.Event e =**

**new ItemChooser.Event(this, selection, values[selection]);**

**// Loop through the listeners using an Iterator**

**for(Iterator i = listeners.iterator(); i.hasNext();) {**

**ItemChooser.Listener l = (ItemChooser.Listener)i.next();**

**l.itemChosen(e); // Notify each listener of the selection**

**}**

**}**

**}**

**// These methods are for event listener registration and deregistration**

**public void addItemChooserListener(ItemChooser.Listener l) {**

**listeners.add(l);**

**}**

**public void removeItemChooserListener(ItemChooser.Listener l) {**

**listeners.remove(l);**

**}**

**/\*\***

**\* This inner class defines the event type generated by ItemChooser objects**

**\* The inner class name is Event, so the full name is ItemChooser.Event**

**\*\*/**

**public static class Event extends java.util.EventObject {**

**int selectedIndex; // index of the selected item**

**Object selectedValue; // the value associated with it**

**public Event(ItemChooser source,**

**int selectedIndex, Object selectedValue) {**

**super(source);**

**this.selectedIndex = selectedIndex;**

**this.selectedValue = selectedValue;**

**}**

**public ItemChooser getItemChooser() { return (ItemChooser)getSource();}**

**public int getSelectedIndex() { return selectedIndex; }**

**public Object getSelectedValue() { return selectedValue; }**

**}**

**/\*\***

**\* This inner interface must be implemented by any object that wants to be**

**\* notified when the current selection in a ItemChooser component changes.**

**\*\*/**

**public interface Listener extends java.util.EventListener {**

**public void itemChosen(ItemChooser.Event e);**

**}**

**/\*\***

**\* This inner class is a simple demonstration of the ItemChooser component**

**\* It uses command-line arguments as ItemChooser labels and values.**

**\*\*/**

**public static class Demo {**

**public static void main(String[] args) {**

**// Create a window, arrange to handle close requests**

**final JFrame frame = new JFrame("ItemChooser Demo");**

**frame.addWindowListener(new WindowAdapter() {**

**public void windowClosing(WindowEvent e) {System.exit(0);}**

**});**

**// A "message line" to display results in**

**final JLabel msgline = new JLabel(" ");**

**// Create a panel holding three ItemChooser components**

**JPanel chooserPanel = new JPanel();**

**final ItemChooser c1 = new ItemChooser("Choice #1", args, null, 0,**

**ItemChooser.LIST);**

**final ItemChooser c2 = new ItemChooser("Choice #2", args, null, 0,**

**ItemChooser.COMBOBOX);**

**final ItemChooser c3 = new ItemChooser("Choice #3", args, null, 0,**

**ItemChooser.RADIOBUTTONS);**

**// An event listener that displays changes on the message line**

**ItemChooser.Listener l = new ItemChooser.Listener() {**

**public void itemChosen(ItemChooser.Event e) {**

**msgline.setText(e.getItemChooser().getName() + ": " +**

**e.getSelectedIndex() + ": " +**

**e.getSelectedValue());**

**}**

**};**

**c1.addItemChooserListener(l);**

**c2.addItemChooserListener(l);**

**c3.addItemChooserListener(l);**

**// Instead of tracking every change with a ItemChooser.Listener,**

**// applications can also just query the current state when**

**// they need it. Here's a button that does that.**

**JButton report = new JButton("Report");**

**report.addActionListener(new ActionListener() {**

**public void actionPerformed(ActionEvent e) {**

**// Note the use of multi-line italic HTML text**

**// with the JOptionPane message dialog box.**

**String msg = "<html><i>" +**

**c1.getName() + ": " + c1.getSelectedValue() + "<br>"+**

**c2.getName() + ": " + c2.getSelectedValue() + "<br>"+**

**c3.getName() + ": " + c3.getSelectedValue() + "</i>";**

**JOptionPane.showMessageDialog(frame, msg);**

**}**

**});**

**// Add the 3 ItemChooser objects, and the Button to the panel**

**chooserPanel.add(c1);**

**chooserPanel.add(c2);**

**chooserPanel.add(c3);**

**chooserPanel.add(report);**

**// Add the panel and the message line to the window**

**Container contentPane = frame.getContentPane();**

**contentPane.add(chooserPanel, BorderLayout.CENTER);**

**contentPane.add(msgline, BorderLayout.SOUTH);**

**// Set the window size and pop it up.**

**frame.pack();**

**frame.setVisible(true);**

**}**

**}**

**}**

**MyFrame.java**

**package SelfMade;**

**import java.awt.event.ActionEvent;**

**import java.awt.event.ActionListener;**

**import java.awt.event.WindowAdapter;**

**import java.awt.event.WindowEvent;**

**import java.awt.\*;**

**import javax.swing.\*;**

**import java.io.File;**

**import java.util.Scanner;**

**import java.util.Vector;**

**import javax.swing.filechooser.FileNameExtensionFilter;**

**public class MyFrame**

**{**

**static JMenuBar menubar;**

**static JFrame frame;**

**static JPanel panel, panell;**

**static FontChooser fontchooser;**

**static JFileChooser filechooser;**

**static JTextArea textarea;**

**static JScrollPane scrollpane1, scrollpane2;**

**static Vector <JLabel> labelvector = new Vector();**

**static JScrollPane scrollpane, scrollpane4;**

**static String Text = "";**

**static Font mainfont;**

**static JMenu FileMenu, ViewMenu, HelpMenu;**

**static JMenuItem Font, Open, Exit, About, LookandFeel;**

**public static void main(String[] args)**

**{**

**frame = new JFrame("My Frame");**

**panel = new JPanel();**

**panell = new JPanel();**

**fontchooser = new FontChooser(frame);**

**filechooser = new JFileChooser();**

**textarea = new JTextArea();**

**textarea.setEditable(false);**

**textarea.setColumns(1);**

**textarea.setLineWrap(true);**

**FileNameExtensionFilter filter = new FileNameExtensionFilter("TEXT FILES", "txt", "text");**

**filechooser.setFileFilter(filter);**

**frame.addWindowListener(new WindowAdapter()**

**{**

**public void windowClosing(WindowEvent e) { System.exit(0); }**

**});**

**menubar = new JMenuBar();**

**frame.setJMenuBar(menubar);**

**FileMenu = createFile(frame);**

**menubar.add(FileMenu);**

**ViewMenu = createView(frame);**

**menubar.add(ViewMenu);**

**HelpMenu = createHelp(frame);**

**menubar.add(HelpMenu);**

**scrollpane = new JScrollPane(textarea);**

**scrollpane.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL\_SCROLLBAR\_ALWAYS);**

**panel.add(scrollpane, BorderLayout.NORTH);**

**scrollpane.setPreferredSize(new Dimension(590, 160));**

**//panell.setAutoscrolls(true);**

**panell.setLayout(new GridLayout(0, 2));**

**for(int i = 0; i<40; i++)**

**{**

**labelvector.add(new JLabel(""));**

**panell.add(labelvector.get(i));**

**}**

**panel.setBackground(new Color(200, 200, 200));**

**panell.setBackground(new Color(0, 200, 200));**

**scrollpane4 = new JScrollPane(panell);**

**scrollpane4.setVerticalScrollBarPolicy(JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS);**

**frame.add(panel, BorderLayout.NORTH);**

**frame.add(panell, BorderLayout.SOUTH);**

**frame.pack();**

**frame.setVisible(true);**

**frame.setSize(600, 700);**

**frame.setResizable(false);**

**}**

**//--------------------------------------------------------------------------**

**public static JMenu createView(final JFrame frame)**

**{**

**JMenu View = new JMenu("View");**

**Font = new JMenuItem("Font");**

**Font.addActionListener(new ActionListener()**

**{**

**public void actionPerformed(ActionEvent e)**

**{**

**fontchooser.setVisible(true);**

**Font font = fontchooser.getSelectedFont();**

**if (font != null)**

**{**

**mainfont = font;**

**FileMenu.setFont(mainfont);**

**ViewMenu.setFont(mainfont);**

**HelpMenu.setFont(mainfont);**

**textarea.setFont(mainfont);**

**Font.setFont(mainfont);**

**Open.setFont(mainfont);**

**Exit.setFont(mainfont);**

**About.setFont(mainfont);**

**LookandFeel.setFont(mainfont);**

**for(int i=0; i<labelvector.size(); i++)**

**labelvector.get(i).setFont(mainfont);**

**textarea.setFont(mainfont);**

**}**

**}**

**});**

**LookandFeel = new JMenu("Look and Feel");**

**ButtonGroup radiogroup = new ButtonGroup();**

**UIManager.LookAndFeelInfo[] plafs = UIManager.getInstalledLookAndFeels();**

**for(int i = 0; i < plafs.length; i++)**

**{**

**String plafName = plafs[i].getName();**

**final String plafClassName = plafs[i].getClassName();**

**JMenuItem item = (JMenuItem) LookandFeel.add(new JRadioButtonMenuItem(plafName));**

**item.addActionListener(new ActionListener()**

**{**

**public void actionPerformed(ActionEvent e)**

**{**

**try**

**{**

**UIManager.setLookAndFeel(plafClassName);**

**SwingUtilities.updateComponentTreeUI(frame);**

**SwingUtilities.updateComponentTreeUI(filechooser);**

**SwingUtilities.updateComponentTreeUI(fontchooser);**

**frame.pack();**

**frame.setSize(600, 400);**

**}**

**catch(Exception ex) { System.err.println(ex); }**

**}**

**});**

**radiogroup.add(item);**

**}**

**View.add(LookandFeel);**

**View.add(Font);**

**return View;**

**}**

**public static JMenu createFile(final JFrame frame)**

**{**

**JMenu View = new JMenu("File");**

**Open = new JMenuItem("Open");**

**Open.addActionListener(new ActionListener()**

**{**

**public void actionPerformed(ActionEvent e)**

**{**

**try**

**{**

**filechooser.show();**

**int res = filechooser.showOpenDialog(null);**

**if(res == 0)**

**{**

**Text = "";**

**File f = filechooser.getSelectedFile();**

**Scanner fin = new Scanner( f );**

**while(fin.hasNextLine())**

**{**

**String str = fin.nextLine();**

**Text+=str + "\n";**

**}**

**String [] stringarray;**

**textarea.setText(Text);**

**stringarray = Text.split("[.!?\n]\\s\*");**

**int counter = 0;**

**for(int i = 0; i<stringarray.length; i++)**

**{**

**if(!stringarray[i].isEmpty())**

**{**

**labelvector.get(counter).setText(stringarray[i]);**

**counter++;**

**}**

**}**

**fin.close();**

**}**

**else { }**

**}**

**catch(Exception ex) { }**

**}**

**});**

**Exit = new JMenuItem("Exit");**

**Exit.addActionListener(new ActionListener()**

**{**

**public void actionPerformed(ActionEvent e)**

**{**

**String ObjButtons[] = {"Yes", "No"};**

**int PromptResult = JOptionPane.showOptionDialog(null,**

**"Are you sure you want to exit?", "Exit",**

**JOptionPane.DEFAULT\_OPTION, JOptionPane.WARNING\_MESSAGE, null,**

**ObjButtons,ObjButtons[1]);**

**if(PromptResult==0)**

**System.exit(0);**

**}**

**});**

**View.add(Open);**

**View.add(Exit);**

**return View;**

**}**

**public static JMenu createHelp(final JFrame frame)**

**{**

**JMenu Help = new JMenu("Help");**

**About = new JMenuItem("About");**

**About.addActionListener(new ActionListener()**

**{**

**public void actionPerformed(ActionEvent e)**

**{**

**String ObjButtons[] = {"OK"};**

**JOptionPane.showOptionDialog(null,**

**"Made by Sheleg Filipp, FAMCS, BSU\n Читает текст из выбранного файла, разбивает его на предложения. В клиентской \n" +**

**"области создать две панели, разделённые по горизонтали, в верхней части отобразить \n" +**

**"содержимое файла, в нижней части отобразить найденные предложения, для каждного\n" +**

**"предложения использовать отдельный компонент JLabel", "About this program",**

**JOptionPane.DEFAULT\_OPTION, JOptionPane.INFORMATION\_MESSAGE, null,**

**ObjButtons,ObjButtons[0]);**

**}**

**});**

**Help.add(About);**

**return Help;**

**}**

**}**

**ShowComponent.java**

**import java.awt.\*;**

**import java.awt.event.\*;**

**import javax.swing.\*;**

**import java.beans.\*;**

**import java.lang.reflect.\*;**

**import java.util.Vector;**

**/\*\***

**\* This class is a program that uses reflection and JavaBeans introspection to**

**\* create a set of named components, set named properties on those components,**

**\* and display them. It allows the user to view the components using any**

**\* installed look-and-feel. It is intended as a simple way to experiment with**

**\* AWT and Swing components, and to view a number of the other examples**

**\* developed in this chapter. It also demonstrates frames, menus, and the**

**\* JTabbedPane component.**

**\*\*/**

**public class ShowComponent {**

**// The main program**

**public static void main(String[] args) {**

**// Process the command line to get the components to display**

**Vector components = getComponentsFromArgs(args);**

**// Create a frame (a window) to display them in**

**JFrame frame = new JFrame("ShowComponent");**

**// Handle window close requests by exiting the VM**

**frame.addWindowListener(new WindowAdapter() { // Anonymous inner class**

**public void windowClosing(WindowEvent e) { System.exit(0); }**

**});**

**// Set up a menu system that allows the user to select the**

**// look-and-feel of the component from a list of installed PLAFs**

**JMenuBar menubar = new JMenuBar(); // Create a menubar**

**frame.setJMenuBar(menubar); // Tell the frame to display it**

**JMenu plafmenu = createPlafMenu(frame); // Create a menu**

**menubar.add(plafmenu); // Add the menu to the menubar**

**// Create a JTabbedPane to display each of the components**

**JTabbedPane pane = new JTabbedPane();**

**// Now add each component as a tab of the tabbed pane**

**// Use the unqualified component classname as the tab text**

**for(int i = 0; i < components.size(); i++) {**

**Component c = (Component)components.elementAt(i);**

**String classname = c.getClass().getName();**

**String tabname = classname.substring(classname.lastIndexOf('.')+1);**

**pane.addTab(tabname, c);**

**}**

**// Add the tabbed pane to the frame. Note the call to getContentPane()**

**// This is required for JFrame, but not for most Swing components**

**frame.getContentPane().add(pane);**

**// Set the frame size and pop it up**

**frame.pack(); // Make frame as big as its kids need**

**frame.setVisible(true); // Make the frame visible on the screen**

**// The main() method exits now but the Java VM keeps running because**

**// all AWT programs automatically start an event-handling thread.**

**}**

**/\*\***

**\* This static method queries the system to find out what Pluggable**

**\* Look-and-Feel (PLAF) implementations are available. Then it creates a**

**\* JMenu component that lists each of the implementations by name and**

**\* allows the user to select one of them using JRadioButtonMenuItem**

**\* components. When the user selects one, the selected menu item**

**\* traverses the component hierarchy and tells all components to use the**

**\* new PLAF.**

**\*\*/**

**public static JMenu createPlafMenu(final JFrame frame) {**

**// Create the menu**

**JMenu plafmenu = new JMenu("Look and Feel");**

**// Create an object used for radio button mutual exclusion**

**ButtonGroup radiogroup = new ButtonGroup();**

**// Look up the available look and feels**

**UIManager.LookAndFeelInfo[] plafs =**

**UIManager.getInstalledLookAndFeels();**

**// Loop through the plafs, and add a menu item for each one**

**for(int i = 0; i < plafs.length; i++) {**

**String plafName = plafs[i].getName();**

**final String plafClassName = plafs[i].getClassName();**

**// Create the menu item**

**JMenuItem item = plafmenu.add(new JRadioButtonMenuItem(plafName));**

**// Tell the menu item what to do when it is selected**

**item.addActionListener(new ActionListener() {**

**public void actionPerformed(ActionEvent e) {**

**try {**

**// Set the new look and feel**

**UIManager.setLookAndFeel(plafClassName);**

**// Tell each component to change its look-and-feel**

**SwingUtilities.updateComponentTreeUI(frame);**

**// Tell the frame to resize itself to the its**

**// children's new desired sizes**

**frame.pack();**

**}**

**catch(Exception ex) { System.err.println(ex); }**

**}**

**});**

**// Only allow one menu item to be selected at once**

**radiogroup.add(item);**

**}**

**return plafmenu;**

**}**

**/\*\***

**\* This method loops through the command line arguments looking for**

**\* class names of components to create and property settings for those**

**\* components in the form name=value. This method demonstrates**

**\* reflection and JavaBeans introspection as they can be applied to**

**\* dynamically created GUIs**

**\*\*/**

**public static Vector getComponentsFromArgs(String[] args) {**

**Vector<Component> components = new Vector<Component>();**

**// List of components to return**

**Component component = null; // The current component**

**PropertyDescriptor[] properties = null; // Properties of the component**

**Object[] methodArgs = new Object[1]; // We'll use this below**

**nextarg: // This is a labeled loop**

**for(int i = 0; i < args.length; i++) { // Loop through all arguments**

**// If the argument does not contain an equal sign, then it is**

**// a component class name. Otherwise it is a property setting**

**int equalsPos = args[i].indexOf('=');**

**if (equalsPos == -1) { // Its the name of a component**

**try {**

**// Load the named component class**

**Class componentClass = Class.forName(args[i]);**

**// Instantiate it to create the component instance**

**component = (Component)componentClass.newInstance();**

**// Use JavaBeans to introspect the component**

**// And get the list of properties it supports**

**BeanInfo componentBeanInfo =**

**Introspector.getBeanInfo(componentClass);**

**properties = componentBeanInfo.getPropertyDescriptors();**

**}**

**catch(Exception e) {**

**// If any step failed, print an error and exit**

**System.out.println("Can't load, instantiate, " +**

**"or introspect: " + args[i]);**

**System.exit(1);**

**}**

**// If we succeeded, store the component in the vector**

**components.addElement(component);**

**}**

**else { // The arg is a name=value property specification**

**String name =args[i].substring(0, equalsPos); // property name**

**String value =args[i].substring(equalsPos+1); // property value**

**// If we don't have a component to set this property on, skip!**

**if (component == null) continue nextarg;**

**// Now look through the properties descriptors for this**

**// component to find one with the same name.**

**for(int p = 0; p < properties.length; p++) {**

**if (properties[p].getName().equals(name)) {**

**// Okay, we found a property of the right name.**

**// Now get its type, and the setter method**

**Class type = properties[p].getPropertyType();**

**Method setter = properties[p].getWriteMethod();**

**// Check if property is read-only!**

**if (setter == null) {**

**System.err.println("Property " + name+**

**" is read-only");**

**continue nextarg; // continue with next argument**

**}**

**// Try to convert the property value to the right type**

**// We support a small set of common property types here**

**// Store the converted value in an Object[] so it can**

**// be easily passed when we invoke the property setter**

**try {**

**if (type == String.class) { // no conversion needed**

**methodArgs[0] = value;**

**}**

**else if (type == int.class) { // String to int**

**methodArgs[0] = Integer.valueOf(value);**

**}**

**else if (type == boolean.class) { // to boolean**

**methodArgs[0] = Boolean.valueOf(value);**

**}**

**else if (type == Color.class) { // to Color**

**methodArgs[0] = Color.decode(value);**

**}**

**else if (type == Font.class) { // String to Font**

**methodArgs[0] = Font.decode(value);**

**}**

**else {**

**// If we can't convert, ignore the property**

**System.err.println("Property " + name +**

**" is of unsupported type " +**

**type.getName());**

**continue nextarg;**

**}**

**}**

**catch (Exception e) {**

**// If conversion failed, continue with the next arg**

**System.err.println("Can't convert '" + value +**

**"' to type " + type.getName() +**

**" for property " + name);**

**continue nextarg;**

**}**

**// Finally, use reflection to invoke the property**

**// setter method of the component we created, and pass**

**// in the converted property value.**

**try { setter.invoke(component, methodArgs); }**

**catch (Exception e) {**

**System.err.println("Can't set property: " + name);**

**}**

**// Now go on to next command-line arg**

**continue nextarg;**

**}**

**}**

**// If we get here, we didn't find the named property**

**System.err.println("Warning: No such property: " + name);**

**}**

**}**

**return components;**

**}**

**}**

**A.12 Исходные коды лабораторной работы №12**

**AddShell.java**

**/\***

**\* To change this license header, choose License Headers in Project Properties.**

**\* To change this template file, choose Tools | Templates**

**\* and open the template in the editor.**

**\*/**

**/\*\***

**\***

**\* @author Angelina**

**\*/**

**public class AddShell {**

**public static native void AddShell1();**

**static{**

**System.loadLibrary("AddShell");**

**}**

**}**

**AddShellTest.java**

**/\***

**\* To change this license header, choose License Headers in Project Properties.**

**\* To change this template file, choose Tools | Templates**

**\* and open the template in the editor.**

**\*/**

**/\*\***

**\***

**\* @author Angelina**

**\*/**

**public class AddShellTest {**

**public static void main(String[] args)**

**{**

**AddShell a = new AddShell();**

**a.AddShell1();**

**}**

**}**

**A.13 Исходные коды лабораторной работы №13**

**ClientMain.java**

**package client;**

**import java.net.MalformedURLException;**

**import java.rmi.\*;**

**import java.rmi.server.\*;**

**import java.rmi.registry.\*;**

**import java.util.Scanner;**

**import csdev.server.ServerInterface;**

**public class ClientMain {**

**public static void main(String[] args) {**

**try {**

**ServerInterface serv = (ServerInterface)Naming.lookup("rmi://127.0.0.1/" + ServerInterface.mudPrefix);**

**System.out.println("Enter your name:");**

**String name;**

**Scanner in = new Scanner(System.in);**

**name = in.next();**

**serv.connect(name);**

**System.out.println("Enter your address:");**

**String address;**

**address = in.next();**

**System.out.println(serv.getMenu());**

**while (true) {**

**System.out.println("Make your order");**

**int choice = 0;**

**choice = in.nextInt();**

**if(serv.takeOrder(choice, address))**

**System.out.println("Order is accepted successfully");**

**else**

**System.out.println("Error: order is not accepted");**

**System.out.println( "Continue ordering? ( 1- Yes, Else - No ) ");**

**int answer = in.nextInt();**

**if ( answer != 1)**

**break;**

**}**

**} catch (Exception e) {**

**// TODO Auto-generated catch block**

**e.printStackTrace();**

**}**

**}**

**}**

**ServerInterface.java**

**package csdev.server;**

**import java.rmi.\*;**

**public interface ServerInterface extends Remote{**

**public String getMenu() throws RemoteException;**

**public boolean takeOrder(int order, String address) throws RemoteException;**

**public boolean connect(String name) throws RemoteException;**

**static final String mudPrefix = "server.orders";**

**}**

**ServerMain.java**

**package csdev.server;**

**import java.rmi.Naming;**

**import java.rmi.RemoteException;**

**import java.rmi.registry.\*;**

**import java.rmi.server.UnicastRemoteObject;**

**public class ServerMain extends UnicastRemoteObject implements ServerInterface {**

**protected ServerMain() throws RemoteException {**

**super();**

**}**

**public static void main(String[] args) {**

**try {**

**LocateRegistry.createRegistry(1099);**

**ServerMain sm = new ServerMain();**

**System.setProperty("java.rmi.server.hostname", "127.0.0.1" );**

**Naming.rebind("//localhost:1099/" + mudPrefix, sm);**

**} catch (Exception e) {**

**e.printStackTrace();**

**}**

**}**

**@Override**

**public String getMenu() throws RemoteException {**

**return "Menu:\n"**

**+ "1-Potatos(0.99$)\n2-Juice(0.09$)\n3-Beer(0.3$)\n"**

**+ "4-Salad(10.99$)";**

**}**

**@Override**

**public boolean takeOrder(int order, String address) throws RemoteException {**

**if (order > 0 && order < 5) {**

**System.out.println("Order taken from address: " + address);**

**return true;**

**}**

**return false;**

**}**

**@Override**

**public boolean connect(String name) throws RemoteException {**

**System.out.println("Connected user: " + name);**

**return true;**

**}**

**}**

**A.14 Исходные коды лабораторной работы №14**

**Servlet.java**

**package com;**

**import javax.servlet.ServletException;**

**import javax.servlet.http.HttpServlet;**

**import javax.servlet.http.HttpServletRequest;**

**import javax.servlet.http.HttpServletResponse;**

**import java.io.File;**

**import java.io.IOException;**

**import java.io.PrintWriter;**

**import java.util.HashMap;**

**import java.util.Scanner;**

**import java.util.StringTokenizer;**

**public class Servlet extends HttpServlet**

**{**

**private volatile String Answer;**

**private volatile int All;**

**private volatile HashMap <String, Integer> Results = new HashMap <String, Integer> ();**

**public void init() throws ServletException**

**{**

**super.init();**

**Answer = "";**

**All = 0;**

**Results.putIfAbsent("C/C++", 0);**

**Results.putIfAbsent("Java", 0);**

**Results.putIfAbsent("C#", 0);**

**Results.putIfAbsent("Python", 0);**

**Results.putIfAbsent("Ruby", 0);**

**}**

**public void toCount(int s)**

**{**

**All++;**

**int a;**

**String lan = "";**

**switch(s)**

**{**

**case 1:**

**lan = "C/C++";**

**break;**

**case 2:**

**lan = "Java";**

**break;**

**case 3:**

**lan = "C#";**

**break;**

**case 4:**

**lan = "Python";**

**break;**

**case 5:**

**lan = "Ruby";**

**break;**

**}**

**a = Results.get(lan);**

**Results.replace(lan, a++);**

**}**

**public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException { performTask(req, res); }**

**public void doPost(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException { performTask(req, res); }**

**public void performTask(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException**

**{**

**res.setContentType("text/html;charset=Windows-1251");**

**res.setCharacterEncoding("Cp1251");**

**req.setCharacterEncoding("Cp1251");**

**PrintWriter out = res.getWriter();**

**Answer = req.getParameter("p0");**

**Answer = Answer.trim();**

**if(Answer.isEmpty())**

**{**

**out.println("<!DOCTYPE HTML><html lang=\"en\"><head><title>Servlet Result</title></head>"**

**+ "<body>"**

**+ "Invalid answer!" +**

**"</body></html>");**

**}**

**else**

**{**

**int ans;**

**try**

**{**

**ans = Integer.getInteger(Answer);**

**if(ans>=1 && ans<=5)**

**toCount(ans);**

**else**

**throw new Exception();**

**out.println("<!DOCTYPE HTML><html><head>");**

**out.println("<meta http-equiv=\"Content-Type\" content=\"text/html; charset=Windows-1251\">");**

**out.println("<title>Servlet Result</title></head>");**

**out.println("<body>");**

**out.println("<h1>Results</h1>");**

**out.println(**

**" <hr><br>\n" +**

**" <p><h2>1) C/C++ - " + Results.get("C/C++")/All\*100 +"</h2>\n" +**

**" <p><h2>2) Java - " + Results.get("Java")/All\*100 +"</h2>\n" +**

**" <p><h2>3) C# - " + Results.get("C#")/All\*100 +"</h2>\n" +**

**" <p><h2>4) Python - " + Results.get("Python")/All\*100 +"</h2>\n" +**

**" <p><h2>2) Ruby - " + Results.get("Ruby")/All\*100 +"</h2>\n" +**

**" <p></p>");**

**out.println("</body></html>");**

**}**

**catch(Exception e)**

**{**

**out.println("<!DOCTYPE HTML><html lang=\"en\"><head><title>Servlet Result</title></head>"**

**+ "<body>"**

**+ "Invalid answer!" +**

**"</body></html>");**

**}**

**}**

**out.close();**

**}**

**}**