Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Кафедра информационных систем управления

ОТЧЕТ

По дисциплине:

«Учебная практика»

Выполнил студент гр. 9

Классковская Надежда

Проверил Высоких Л. К.

Минск 2015

СОДЕРЖАНИЕ

1 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №1 3

2 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №2 4

3 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №3 6

4 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №4 8

5 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №5 10

6 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №6 12

7 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №7 13

8 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №8 14

9 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №9 16

10 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №10 18

11 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №12 19

12 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №13-14 21

13 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №15-16 23

**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**

**Постановка задачи:**

* Создать класс, реализующий интерфейс Shape, для изображения шара в двумерном пространстве;
* С помощью аффинного преобразования координат изобразить приближение
* либо удаление шара. Фигура перемещается с постоянной скоростью;
* Рисунок выполнить в окне апплета или фрейма с выбранной толщиной границы фигуры, цветом границы и цветом внутренней области (толщина и цвет вводятся в качестве аргументов программы или параметров апплета).

**Решение задачи:**

Для отрисовки круга создается класс-наследник класса Ellipse2D.Double, который в свою очередь реализует интерфейс Shape. Добавляется метод draw, отвечающий за непосредственную отрисовку круга. Также используются дополнительные поля, отвечающие за скорость движения шара и изменение масштаба при аффинных преобразованиях. В апплете используется специальный парсер, считывающий шестнадцатеричные коды цветов. Для плавности работы рисунок выполняется на панели.

**Результат:**

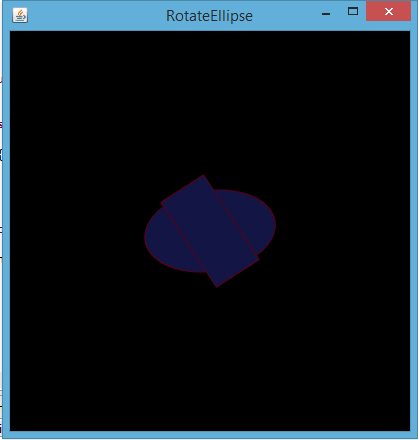


Рисунок 1.1 – Скриншот окна приложения

**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**

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

**Предметная область:**

Двумерная графика.

**1 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

* Для обработки изображений в Java 2D API введены следующие классы и интерфейсы:
* Интерфейс Paint, описывающий заливку фигур. Этот интерфейс реализуют классы Color, что определяет сплошную заливку, GradientPaint, описывающий градиентную заливку, и TexturePaint, который заполняет фигуру текстурой.
* Интерфейс Composite, описывающий композиции цветов. AlphaComposite является одним из классов, реализующих этот интерфейс. Он комбинирует цвета, основываясь на их прозрачности (значении альфа-канала).
* Класс RenderingHints для настройки параметров отрисовки как, например, сглаживание.
* Класс BufferedImage, для работы с внеэкранными буферами.
* Интерфейс BufferedImageOp, при помощи которого определяются фильтры обработки изображений
  1. **ЗАДАНИЕ**

1. **Постановка задачи**

Изобразить дорожный знак: вопросительный знак в квадрате, цвет квадрата и символа

– желтый, цвет фона – серый с градиентной заливкой слева-направо.

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**

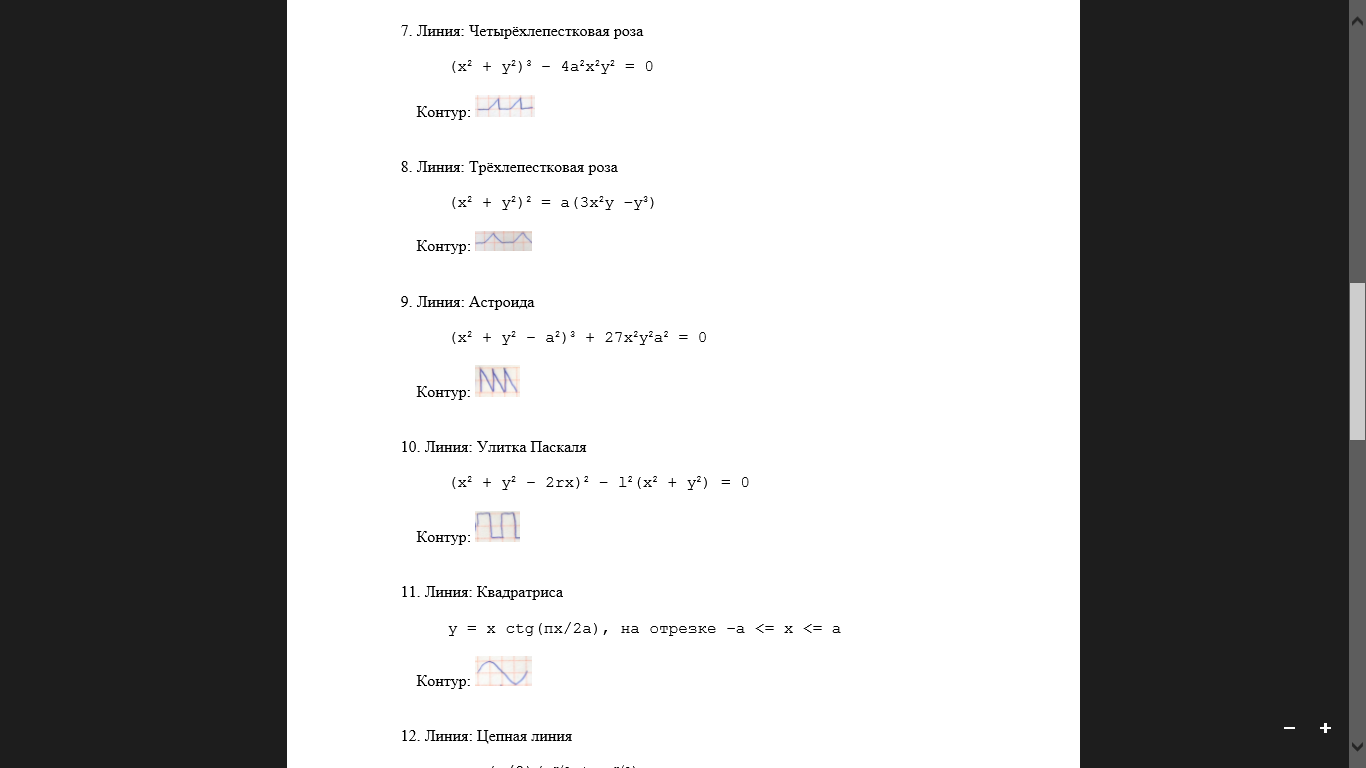
Класс Stroke преобразует операцию рисования линий в операцию заливки области, принимая объект Shape, очертания которого нужно прорисовать, и возвращая штриховую фигуру, представляющую собой контур исходной фигуры. Поскольку интерфейс Stroke прост, то относительно нетрудно определять собственные классы,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

**Результат:**

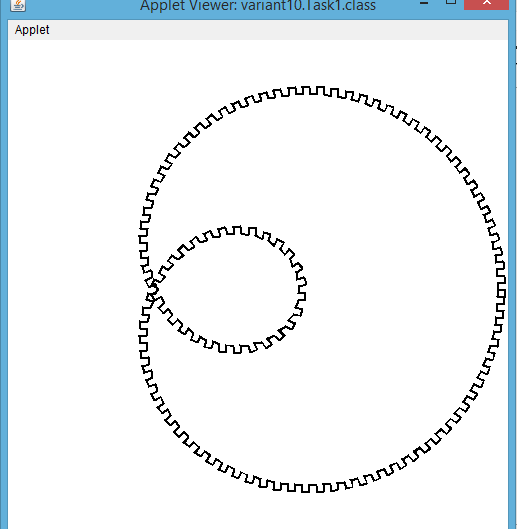


Рисунок 3.1 – Результат работы приложения

**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**

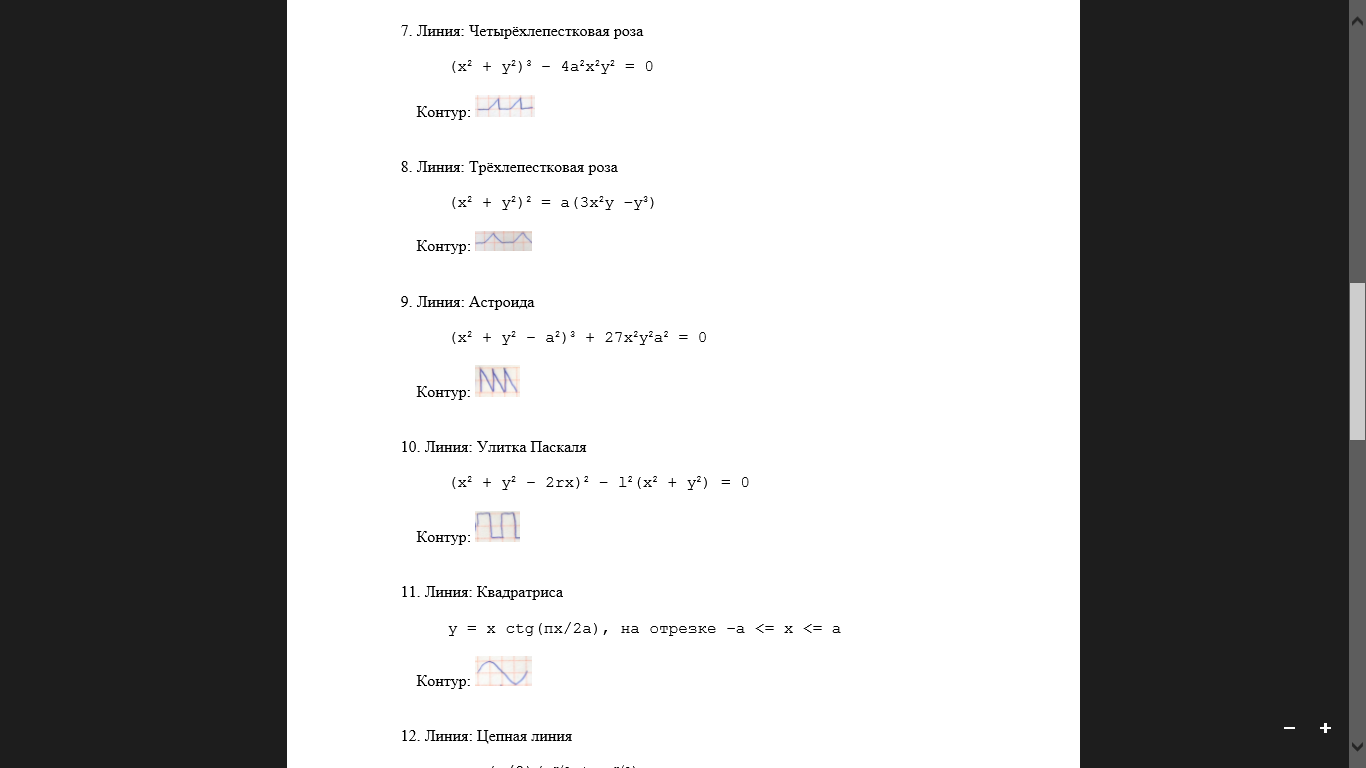
Для печати документов в Java используются несколько основных классов. Это класс PrinterJob, который управляет печатью. С его помощью можно вызвать диалоговое окно печати с пользователем и настроить соответствующие свойства.

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**

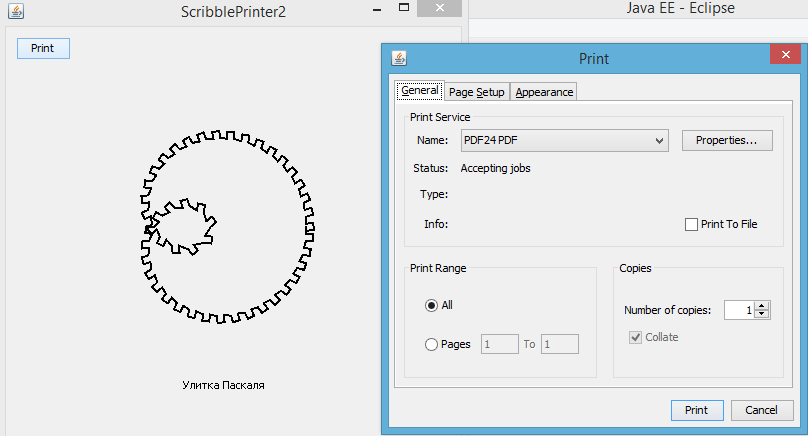


Рисунок 4.1 – Процесс работы приложения

**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**

**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).

**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.

**Результат:**

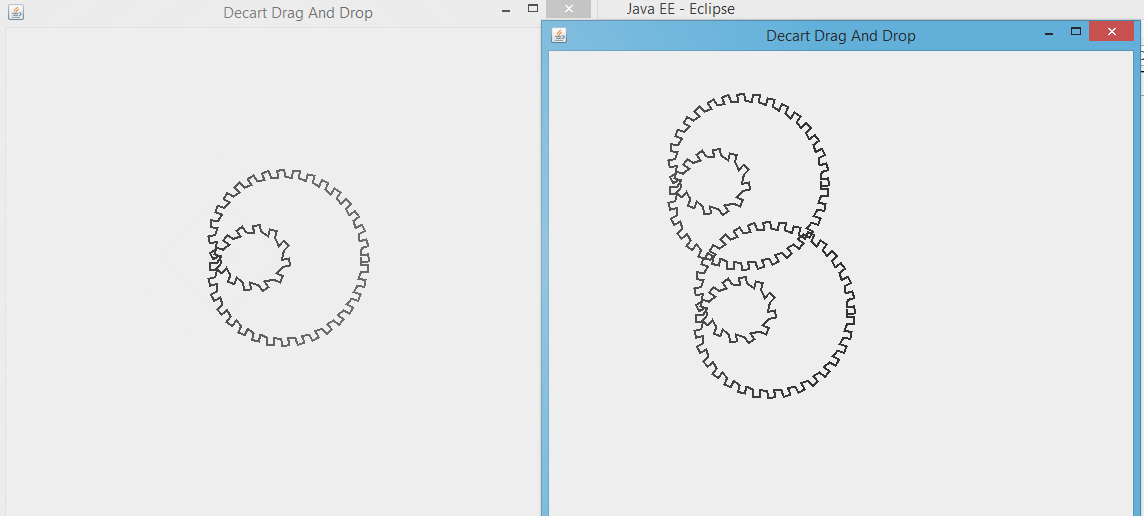


Рисунок 5.1 – Вид итогового приложения

**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 несколькими.

**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.

**2** **Задание**

**Постановка задачи:**

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).

**Решение задачи:**

Данные о каждом монументе хранятся в объектах класса *ScheduleLeaf*. Сами эти объекты находятся внутри *ScheduleTableModel* в виде массива. *ScheduleModel* является моделью как для дерева, так и для таблицы.

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.

**Результат:**

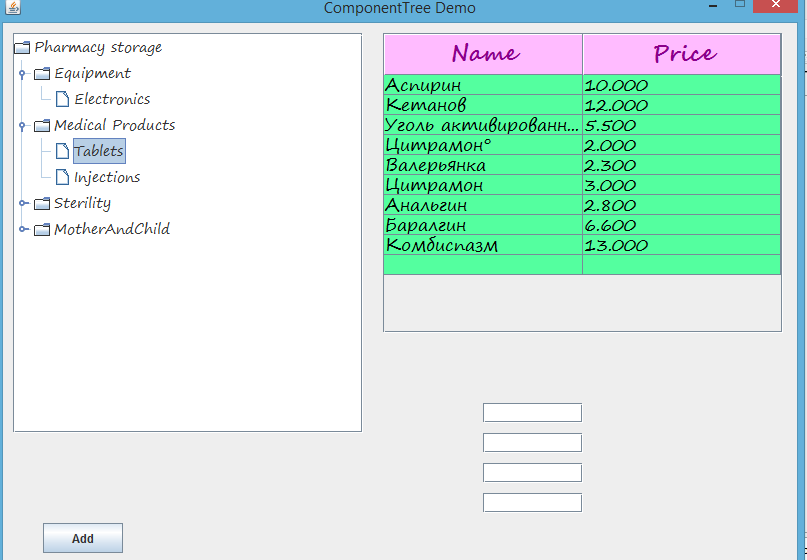


Рисунок 6.1 – Результат работы программы

**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 *TableMo7del* 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 «плоская вертикальная линия» на базе класса Canvas. Свойства: высота, цвет. Создать файл манифеста и упаковать компонент вместе с исходным кодом разработанных классов.

**2.2 Решение задачи**

Для представления вертикальной линии был создан класс VerticalLine. 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: VerticalLine.class Java-Bean: True

Созданный компонент упаковывается в .jar-архив (Lab7.jar). Чтобы использовать его, нужно установить его в JavaBeans.

**2.3 Результат:**

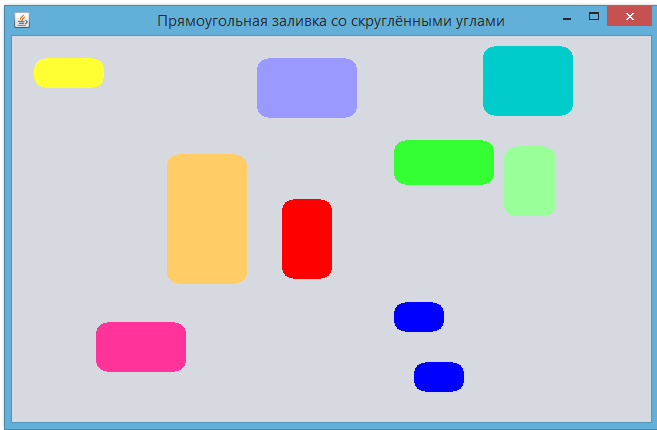


Рисунок 7.1 – Запуск фрейма

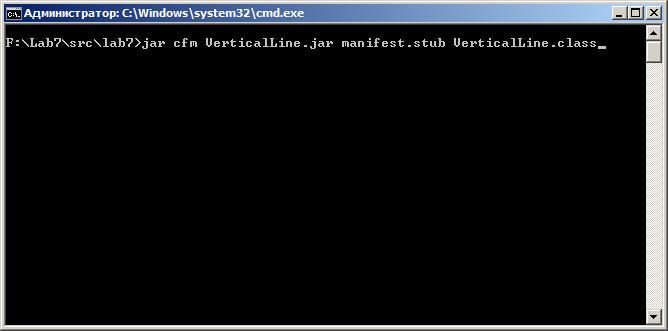


Рисунок 7.2 – Использование файла манифеста для создания .jar-архива

* 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**

**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.

**Решение задачи:**

В качестве компонента *JavaBeans* выступает класс My*Panel*. 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 *MyPanelBeanInfo*. Большая его часть была автоматически сгенерирована средой

*NetBeans*.

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

**Результат:**

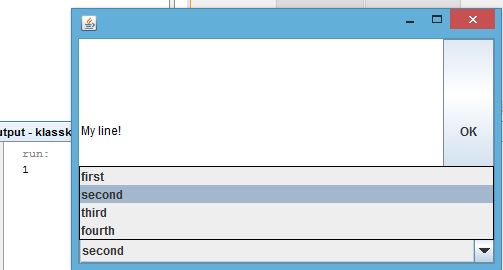


Рисунок 8.1 – Использование компонента в *NetBeans*

**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. Если ваш тип определен в классе X, редактор для него должен быть определен в классе XEditor.

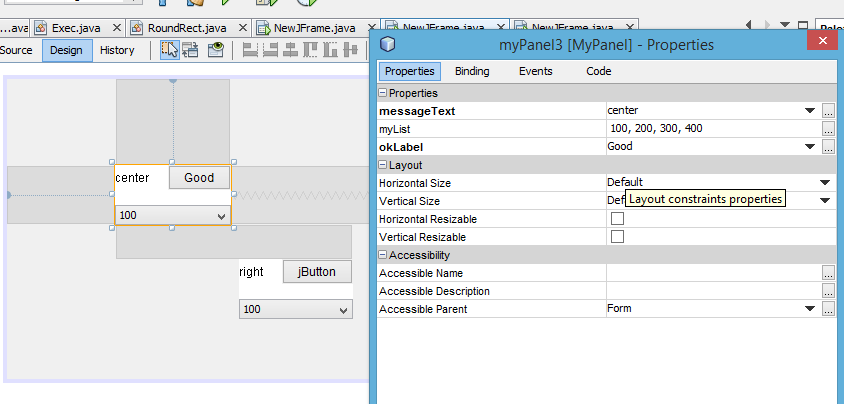
Другим возможным способом является регистрация редактора свойств путем вызова метода PropertyEditorManager.registerEditor(), например, из конструктора вашего класса BeanInfo. В случае если вы будете вызывать этот метод из самого компонента, компонент будет зависеть от класса редактора свойств, поэтому 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.

* 1. **ЗАДАНИЕ**

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.

* 1. **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.
  + 1. **Результат:**



* 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:**

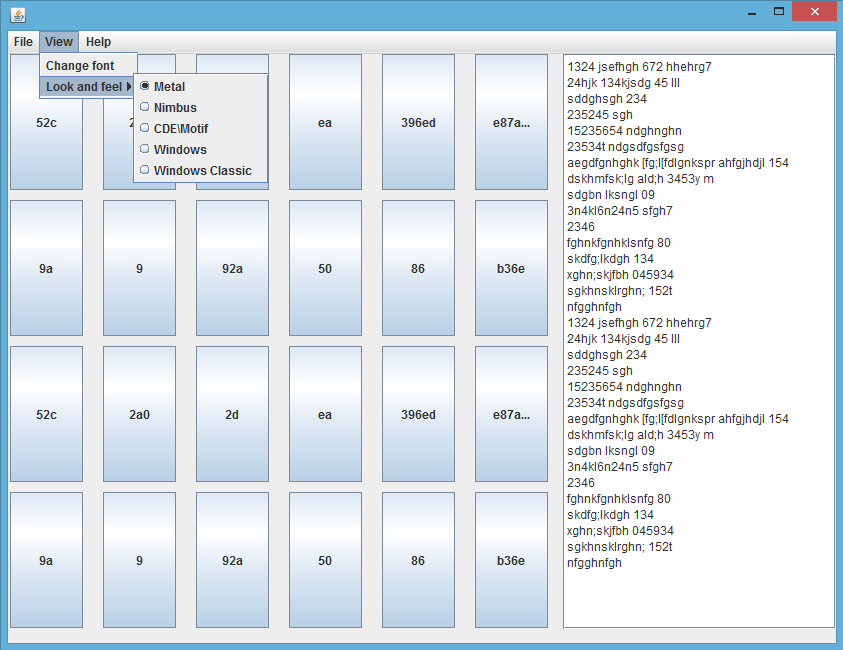


Рисунок 10.1 – Результат работы приложения

**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.

* 1. **ЗАДАНИЕ**

1. **Постановка задачи**

Разработать класс для добавления и удаления иконок в панель задач. Использовать функцию Shell\_NotifyIcon. Продемонстрировать решение. Для разработки библиотеки использовать Win32 API и MS Visual Studio C/C++.

**2.2 Решение задачи**

При решении используется класс IconLibrary, в котором объявляются два статических нативных метода: на добавление иконки и на удаление иконки. Методы принимают идентификатор иконки – с каким идентификатором создадим иконку, с тем нужно будет и работать при удалении. Библиотека загружается в классе в статическом блоке. Используется при этом функция System.LoadLibrary (string). Функция грузит dll, а затем можно свободно вызывать подгруженные методы. В библиотеке функции объявляются в .h файле, определяются – в .cpp файле.

**2.3 Результат:**

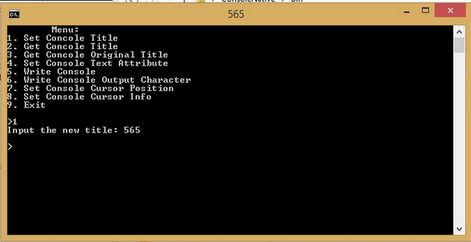


Рисунок 11.1 – Работа с командами консоли

**4 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.
* Иконка создается методом Shell\_NotifyIcon. У этого метода очень широкие возможности. Иконку можно всячески настроить, реагируя на события левой кнопки, правой, перетаскивания и т.д. Делать такие вещи в Win32 API однозначно удобнее, поэтому Java использует именно Win32 API через нативные методы, а не собственный API. Даже серьезный недостаток, заключающийся в непортируемости, не играет такой серьезной роли.

**12 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №13-14**

**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:**

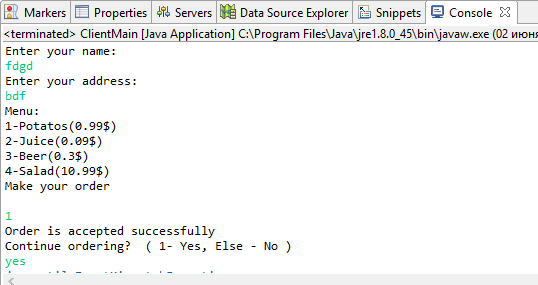


Рисунок 12.1 – Результат работы программы (создание сервера, добавление клиентов)

**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 №15-16**

**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++).

* + 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**

**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.

**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/

Фрагмент разобранного кода:

import java.io.\*;

import java.sql.\*;

import java.util.ArrayList;

import java.util.Map;

import java.util.Vector;

import javax.servlet.ServletException;

import javax.servlet.http.\*;

public class Register extends HttpServlet {

public ArrayList <User> users=new ArrayList<User>();

static int next\_id=0;

public void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String n=request.getParameter("First name");

User user=new User(next\_id++, n, p, e, c, d);

Boolean flag=true;

for(int i=0; i<users.size(); i++)

{ if(users.get(i).compareTo(user)==0)

}}

**Результат:**



Рисунок 13.1 – пример сервлета TomCat

**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.