МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики
Кафедра информационных систем управления

ОТЧЕТ
По дисциплине:
«Учебная практика»

Выполнил
студент 2 курса 9 группы
Шелег В. М.

Проверил
Высоких Л.К.

Минск 2017

Содержание

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.

1.1 Краткие теоретические сведения

Средства двумерной графики в Java включают в себя следующее:

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

1.2 Задание

Постановка задачи:

Изобразить пятиконечную звезду, вписанную в окружность. Задать вращение фигуры вокруг выбранной точки по часовой стрелке.

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

Решение задачи:

Для создания звезды в круге был создан класс, реализующий интерфейс Shape.

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

Для задания точки вращения и позиции фигуры используется параметр апплета.

В программу передается 7 параметров, которые отвечают за местоположение фигуры, за размеры круга и вписанной в него звезды, за толщину и цвет пера, а также за цвет заливки.

Результат:

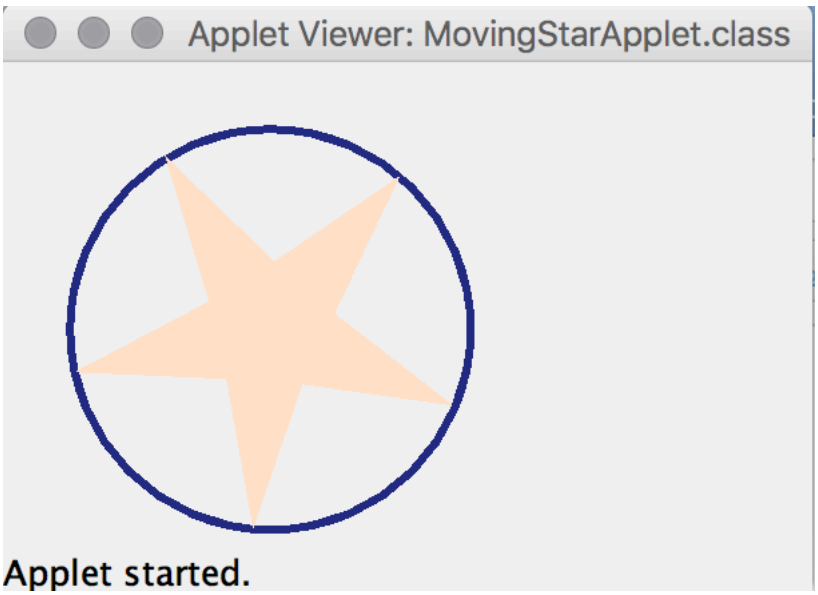

Рис. 1.1 – Приложение во время запуска

1.3 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.
- 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.

2.1 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

2.2 Задание

Постановка задачи:

Фигура (дорожный знак): надпись GO в прямоугольнике, цвет прямоугольника и надписи – синий, цвет фона – серый с градиентной заливкой сверху-вниз Фильтр: Mirror image

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.

Решение задачи:

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

Тень от знака рисуется при помощи аффинных преобразований над исходным знаком. Тень полупрозрачна.

В качестве фильтра используется объект класса AffineTransformOp. Для градиентной заливки используется класс GradientPaint.

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

Результат:

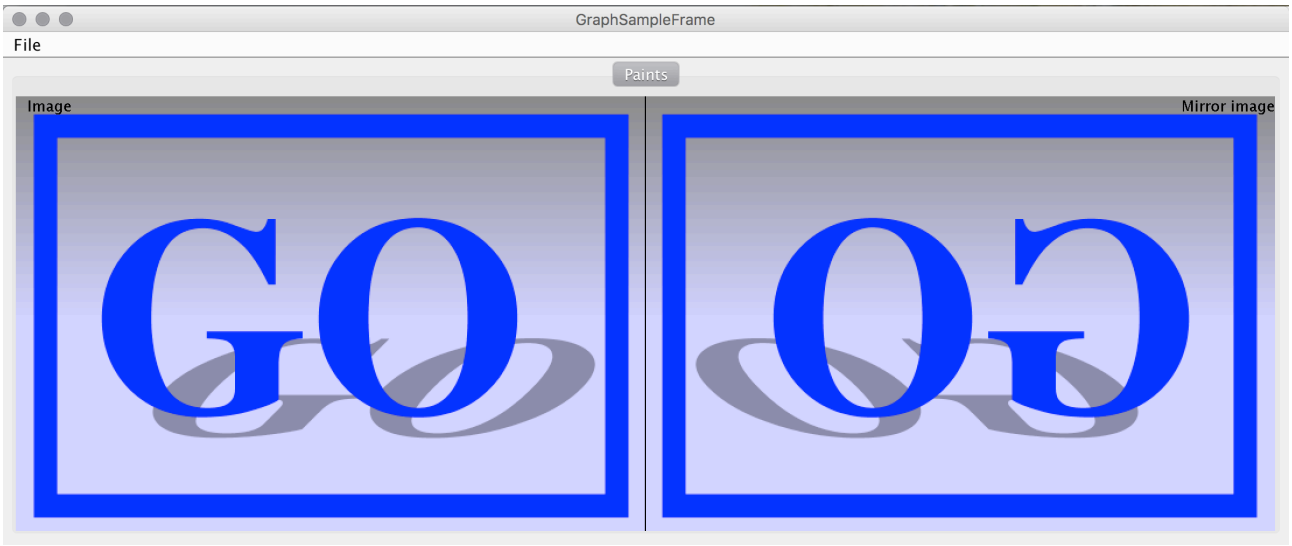

Рис. 2.1 – Результат работы приложения

2.3 Выводы

- Работа с внеэкранными буферами незначительно отличается от работы напрямую. Необходимо только воспользоваться методом BufferedImage.getGraphics(), после чего работать с привычным классом Graphics.
- Для установки сглаживания необходимо установить ключи ANTIALIASING и TEXT_ANTIALIASING в объекте класса RenderHints.
- Множество фильтров обработки изображений уже реализованы в Java и они достаточно просты в использовании.

3 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №3

Цель:

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

Предметная область:

Двумерная графика.

3.1 Краткие теоретические сведения

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

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

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

3.2 Задание

Постановка задачи:

Разработать пользовательский класс Shape реализующий рисование квадратирисы. Разработать пользовательский класс Stroke для отображения зубчатого с промежутками контура. Создать приложение или апплет для тестирования и демонстрации разработанных классов.

Решение задачи:

Для отрисовки локона Аньези был создан класс WitchOfAgnesi, который реализует интерфейс Shape. Координаты высчитываются приближенно.

За характер контура отвечает созданный класс ToothStroke, реализующий интерфейс Stroke. В методе createStrokedShape на основании заданной фигуры рисуется новая с соответствующим контуром при помощи класса FlatteningPathIterator.

Результат:

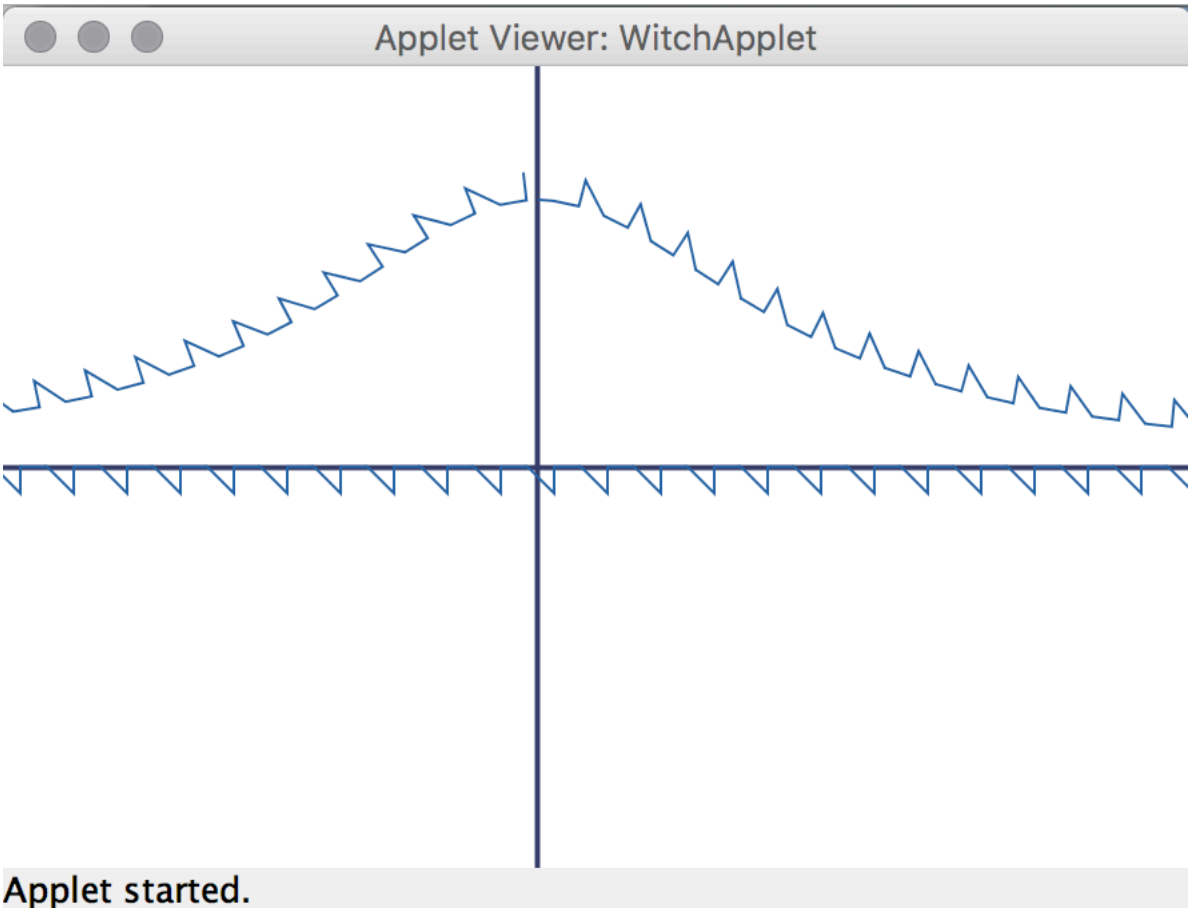

Рис. 3.1 – Результат работы приложения

3.3 Выводы

- Основной задачей при создании своей фигуры является созданием для нее PathIterator’a. Именно он отвечает за вид фигуры.
- При создании своей фигуры можно рисовать не только прямыми линиями, но также и кривыми Безье (SEG_CUBICTO, SEG_QUADTO).
- Процесс создания своей кисти не сильно отличается от создания PathIterator’a для фигуры, за исключением того, что нужно ориентироваться на уже существующую фигуру.

4 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №4

Цель:

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

Предметная область:

Двумерная графика, редактирование текста.

4.1 Краткие теоретические сведения

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

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

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

4.2 Задание

Постановка задачи:

Для выполнения задания используется ваш вариант решения задания №3.

Модифицируйте программу из предыдущей лабораторной работы следующим образом. В демонстрационное приложение добавить возможность печати небольшого отчёта о решении задания №3. Отчёт должен содержать следующее:

- Рисунок с подписью алгебраической линии задания
- Исходный текст класса Shape, реализующий рисование указанной алгебраической линии.

При печати использовать режим альбомной ориентации страницы и двустороннюю печать.

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

Решение задачи:

Для печати была добавлена кнопка, которая по нажатию выводит диалог PrintJob’a и печатает отчет. Печать отчета представляет собой вызов print для объекта класса MainFrame, который реализует Printable..

После буфера на странице печатается подпись и начинается печать исходного текста. Текст хранится как массив строк. При печати разрывы страниц вычисляются вручную.

Результат:

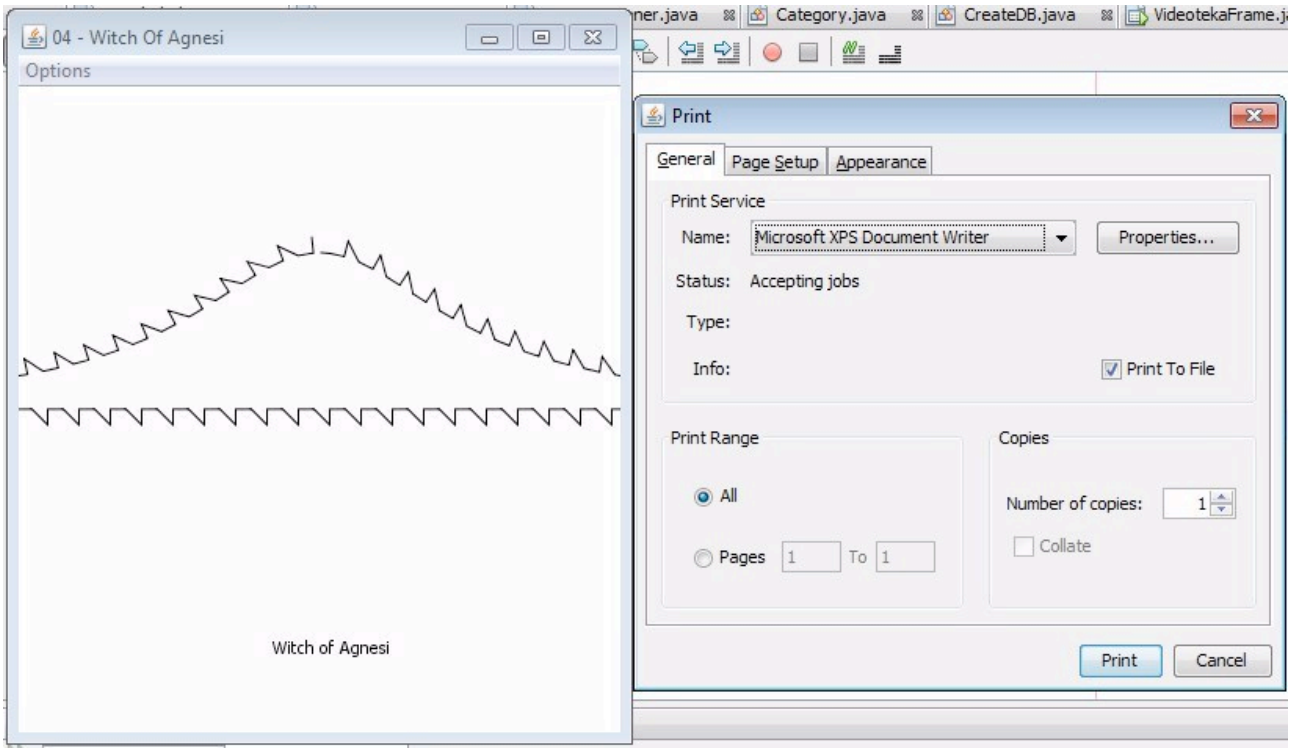

Рис. 4.1 – Вид первой печатной страницы

4.3 Выводы

- Работа с принтерами обеспечивается классом PrinterJob, в котором уже есть готовый графический интерфейс для настройки печати.
- Для того, чтобы узнать, сколько текст будет занимать места на странице можно использовать класс FontMetrics.
- Если текст не помещается на одну страницу, то разбивку на страницы необходимо делать самостоятельно.

5 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №5

Цель:

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

Предметная область:

Двумерная графика, разработка графических интерфейсов.

5.1 Краткие теоретические сведения

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

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

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

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

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

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

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

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

5.2 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 *WitchOfAgnesi* 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.

Drag-and-drop работает как в пределах одного приложения, так и в пределах нескольких.

Результат:

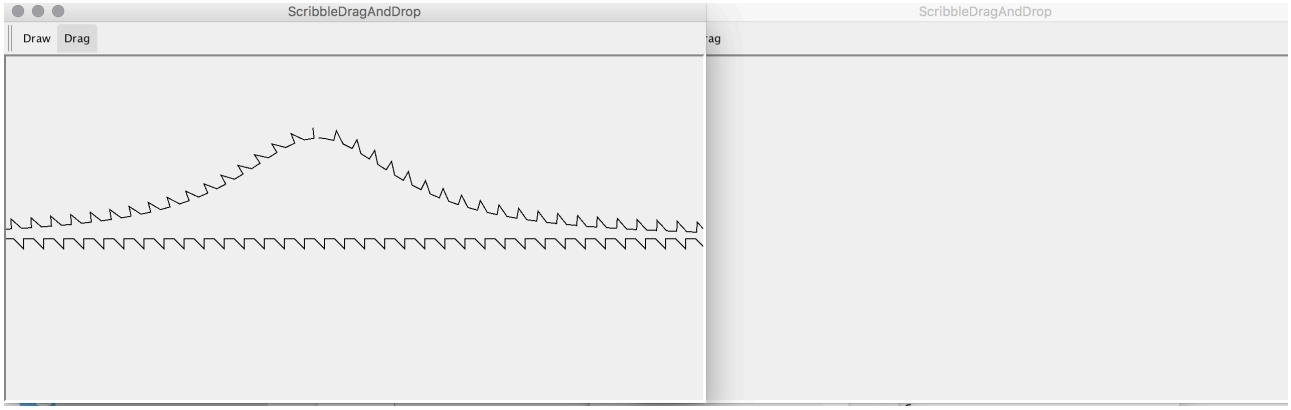

Рис. 5.1 – Вид итогового приложения

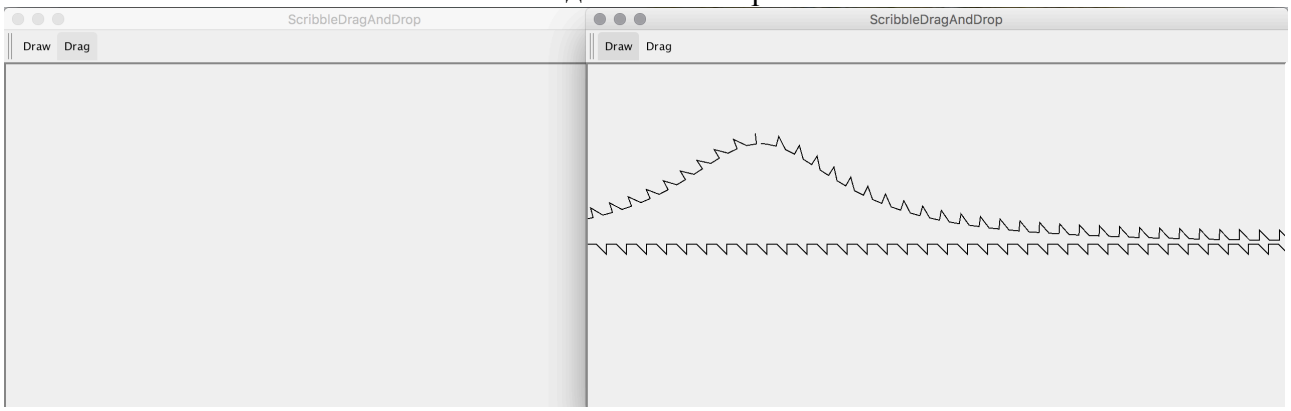

Рис. 5.2 – Вид после перемещения

5.3 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 комнатных растений.

6.1 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 *JTree*, 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*.

6.2 Задание

Постановка задачи:

Разработать систему классов/интерфейсов для каталога комнатных растений. Данные необходимо упорядочить по атрибутам/свойствам товаров, предметов и т.п. в виде дерева.

Разработать графическое приложение для ввода/отображения данных комнатных растений. 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 *FlowerNode*. Сами эти объекты находятся внутри *Catalog* в виде массива.

На главной форме находятся дерево, при помощи которого можно осуществлять фильтрацию монументов, 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 монументов.

Через пункт меню *Добавить цветок в каталог* можно добавить новый цветок в базу. Открывается новое окно, в котором вводятся данные при помощи элементов *HintTextBox* – наследник от *JTextBox*, который поддерживает отображение подсказок. Удалить выделенную запись можно при помощи пункта меню *Удалить цветок из каталога*.

Результат:

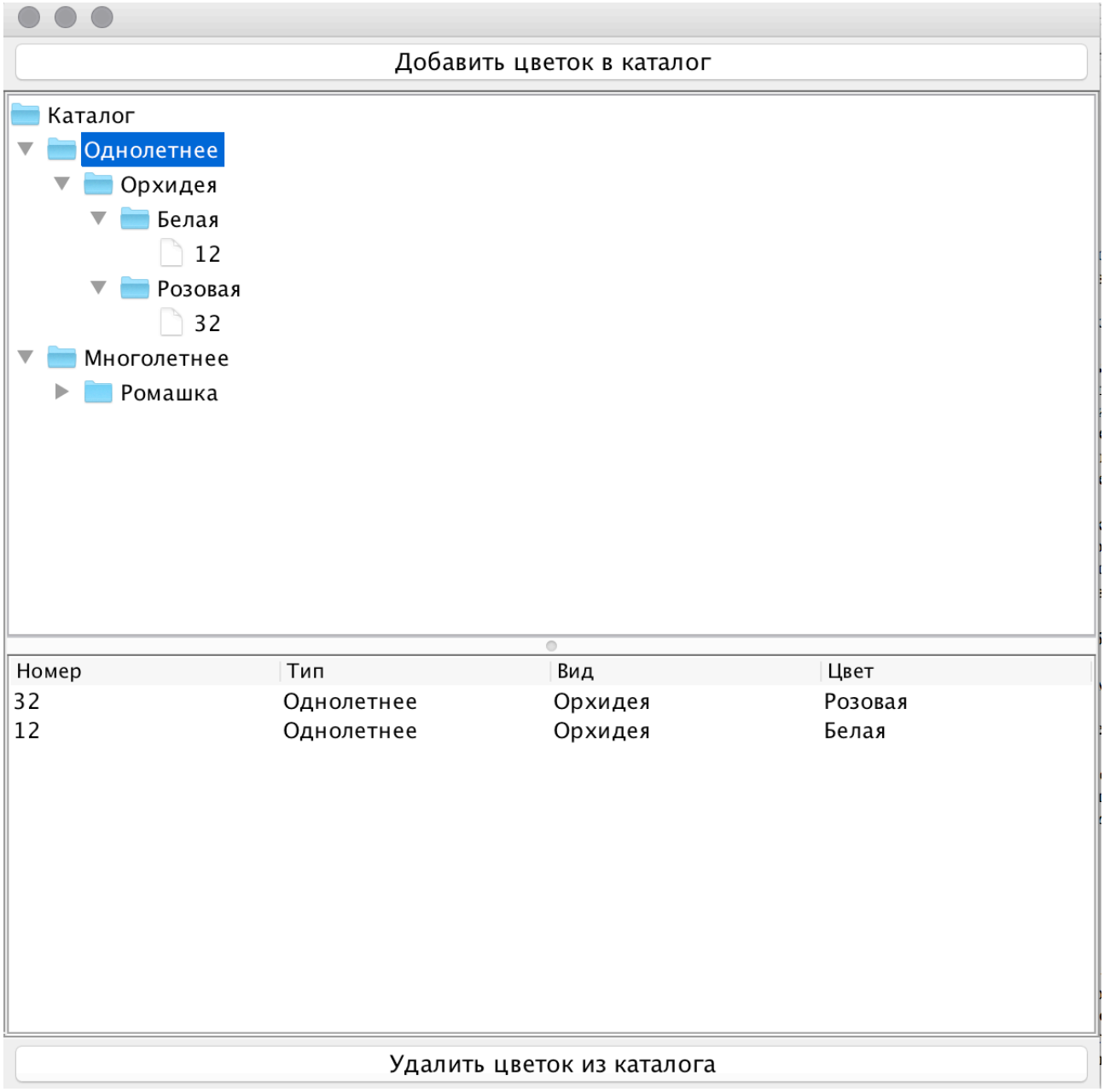

Рис. 6.1 – Вид главного окна

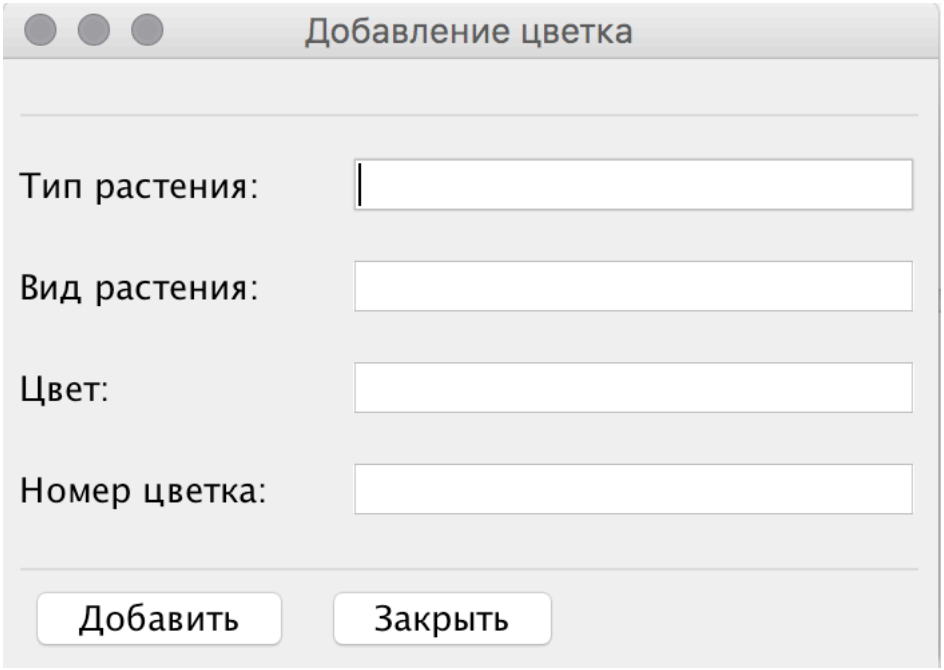

Рис. 6.2 – Вид окна добавления нового цветка

6.3 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 разработке своих собственных БД.

Предметная область:

Архитектура БД.

7.1 Краткие теоретические сведения

Реляционная модель данных. Основная идея этой модели – представить любой набор данных в виде двумерной таблицы. Класс java.sql.Connection представляет в JDBC сеанс работы с базой данных. Он предоставляет приложению объекты Statement (и его подклассы) для этого сеанса. Он также управляет транзакциями для этих команд. По умолчанию каждая команда выполняется в отдельной транзакции. Основной метод класса Connection createStatement() используется для объекта Statement, связанного с сеансом Connection. Класс Statement представляет встроенную команду SQL и используется приложением для доступа к базе данных. При закрытии Statement автоматически закрываются все связанные с ним открытые объекты ResultSet. Основные методы для работы с классом Statement:

- ResultSet executeQuery(String sql) используется для выполнения запросов (на извлечение данных). Он возвращает для обработки результирующий набор.
- int executeUpdate(String sql) используется для выполнения обновлений. Он возвращает количество обновленных строк.
- Интерфейс PreparedStatement наследует от Statement и отличается от последнего следующим:
- Экземпляры PreparedStatement "помнят" скомпилированные SQL-выражения. Именно поэтому они называются "prepared" ("подготовленные").
- SQL-выражения в PreparedStatement могут иметь один или более входной (IN) параметр. Входной параметр - это параметр, чье значение не указывается при создании SQL-выражения. Вместо него в выражении на месте каждого входного параметра ставится знак ("?"). Значение каждого вопросительного знака устанавливается методами setXXX перед выполнением запроса.

Класс ResultSet представляет результирующий набор базы данных. Он обеспечивает приложению построчный доступ к результатам запросов в базе данных. Во время обработки запроса ResultSet поддерживает указатель на текущую обрабатываемую строку. Приложение последовательно перемещается по результатам, пока они не будут все обработаны или не будет закрыт ResultSet.

7.2 Задание:

Постановка задачи:

Исследовать предложенную предметную область, спроектировать структуру базы

данных объектов выбранной предметной области (из не менее чем 2-х таблиц объектов). Согласуйте проект БД с преподавателем.

- Разработайте графическое приложение для создания/ввода/отображения БД Вашего варианта задания. Содержимое БД отображайте в виде таблиц. - При реализации интерфейса следуйте рекомендациям стандарта CUI (Common User Interface).

Решение задачи:

В качестве реализации работы с базой данных используется класс BD, наследующий класс Application для корректного отображения данных, полученных с нашей БД. Для работы с базой данных будем использовать БД Sqlite3.

Результат:

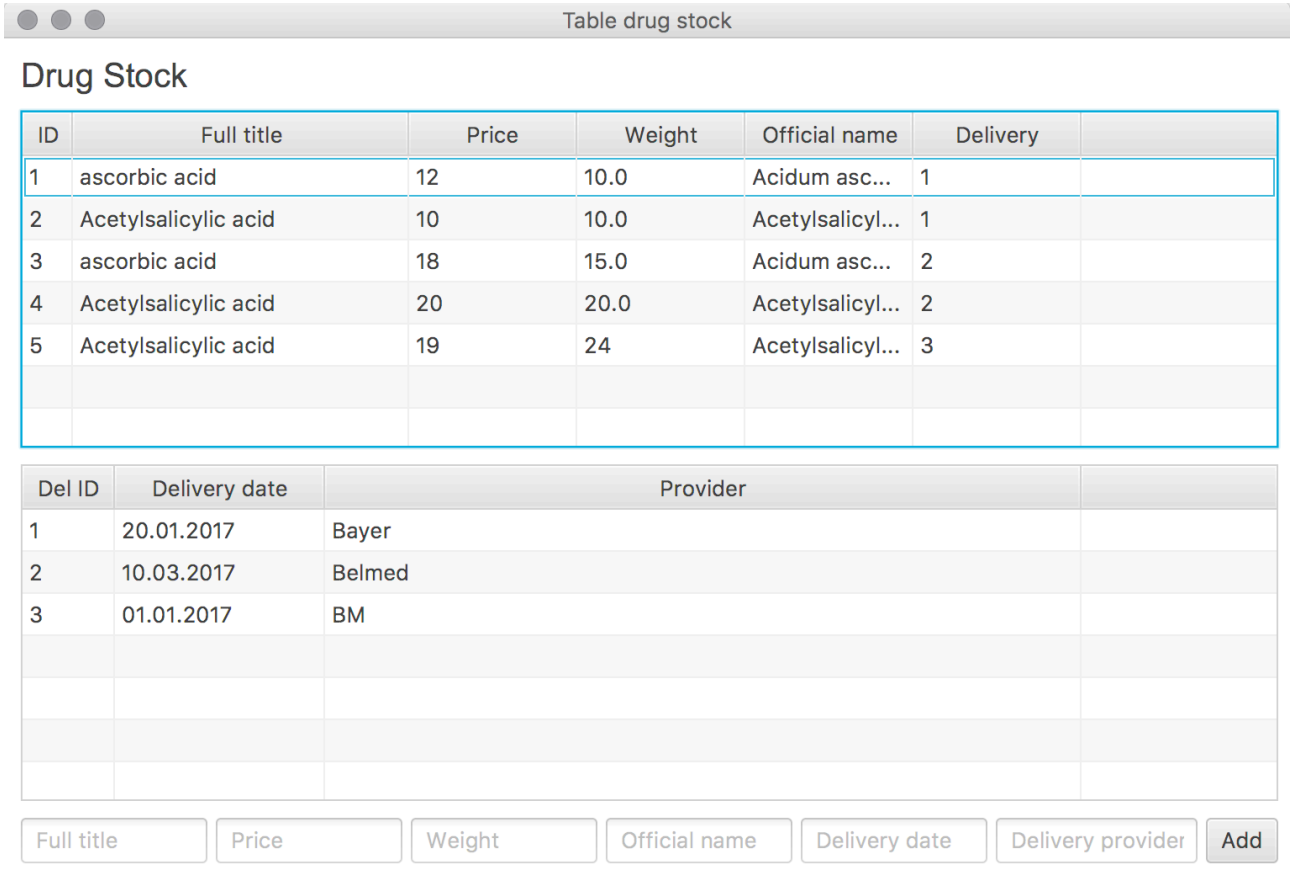

Рис. 7.1 – Главное окно

7.3 Выводы

- Для того, чтобы корретно отображать данные можно использовать Java FX.
- Существует множество вариантов БД, однако наиболее приятная в использовании для знакомства с БД есть Sqlite3.

8 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №8

Цель:

Научиться разработке своих собственных компонентов *JavaBeans*.

Предметная область:

Графический интерфейс пользователя.

8.1 Краткие теоретические сведения

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

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

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

Класс должен:

- Класс должен иметь конструктор без параметров, с модификатором доступа *public*. Такой конструктор позволяет инструментам создать объект без дополнительных сложностей с параметрами.
- Свойства класса должны быть доступны через *get*, *set* и другие методы (так называемые методы доступа), которые должны подчиняться стандартному соглашению об именах. Это легко позволяет инструментам автоматически определять и обновлять содержание bean’ов. Многие инструменты даже имеют специализированные редакторы для различных типов свойств.
- Класс должен быть сериализуем. Это даёт возможность надёжно сохранять, хранить и восстанавливать состояние bean независимым от платформы и виртуальной машины способом.

8.2 Задание

Постановка задачи:

Разработать простой компонент на базе класса *Canvas* – плоскую горизонтальную линию. Создать файл манифеста и упаковать компонент вместе с исходным кодом разработанных классов. При разработке поместить все классы в пакет *bsu.fpmi.educational_practice*.

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

Решение задачи:

В качестве компонента *JavaBeans* выступает класс *HorizontalLine*. В нем присутствуют свойства *get/setLineWidth* и *get/setLineColor*, которые отвечают за ширину и цвет линии соответственно. Класс унаследован от класса *Canvas*.

Был создан jar архив для интеграции компонента в *NetBeans*. Файл манифеста для его создания содержал строки:

Name: HorizontalLine.class

Java-Bean: true

Результат:

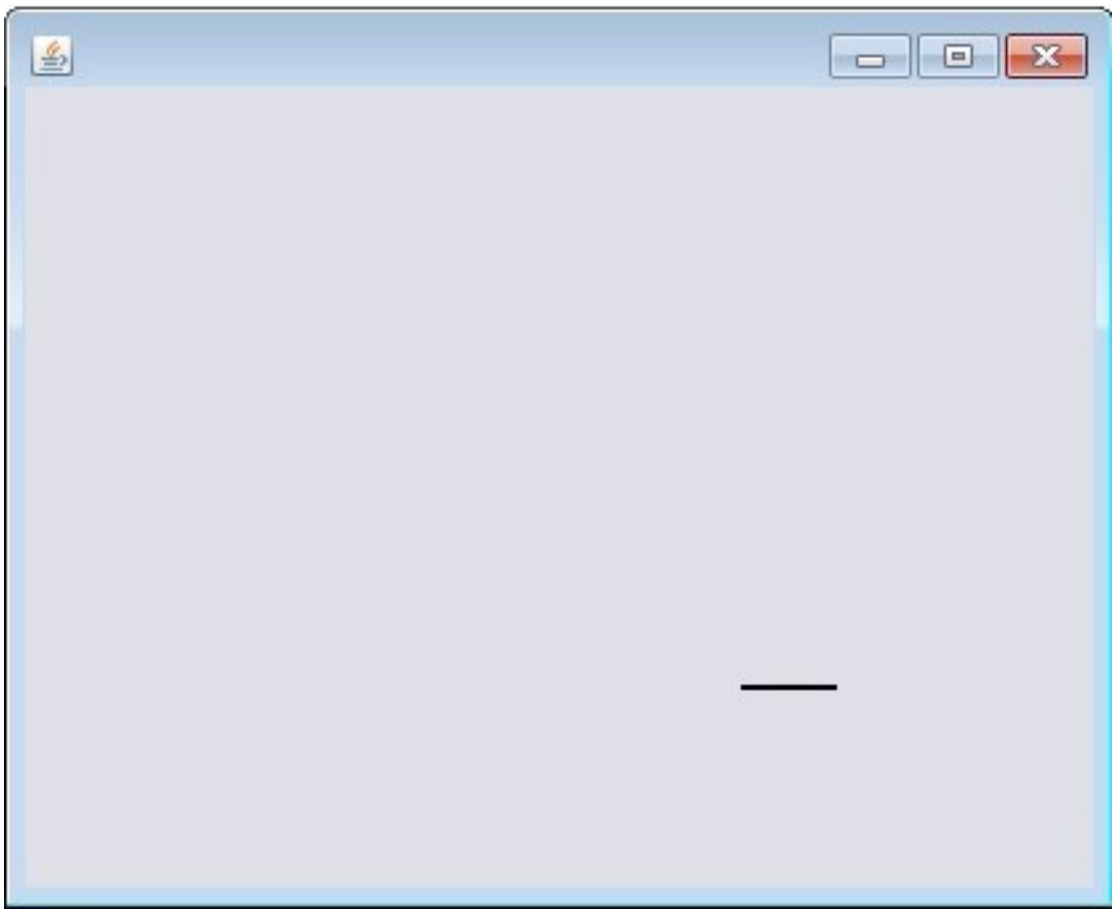

Рис. 8.1 – Главное окно

8.3 Выводы

- Для того, чтобы класс соответствовал спецификации *JavaBeans*, он просто должен реализовывать интерфейс *Serializable*, иметь публичный конструктор по умолчанию и для свойств иметь соответствующие методы *get* и *set*.
- Реализация компонента как *JavaBeans* компонента позволяет его легко использовать в визуальных редакторах интерфейсов, например, в *NetBeans*.

9 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №9

Цель:

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

Предметная область:

Графический интерфейс пользователя.

9.1 Краткие теоретические сведения

Схема 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.

9.2 Задание

Постановка задачи:

Разработать компонент: две независимые радио-кнопки, лейбл и кнопка, событие при нажатии на кнопку. Создать файл манифеста и упаковать компонент. При разработке поместить все классы в пакет *bsu.fpmi.educational_practice*. 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 *AcceptEvent*. Определить интерфейс слушателя *AcceptListener*. 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 *MyComp* 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 *MyCompBeanInfo*. Большая его часть была автоматически сгенерирована средой *NetBeans*.

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

Результат:

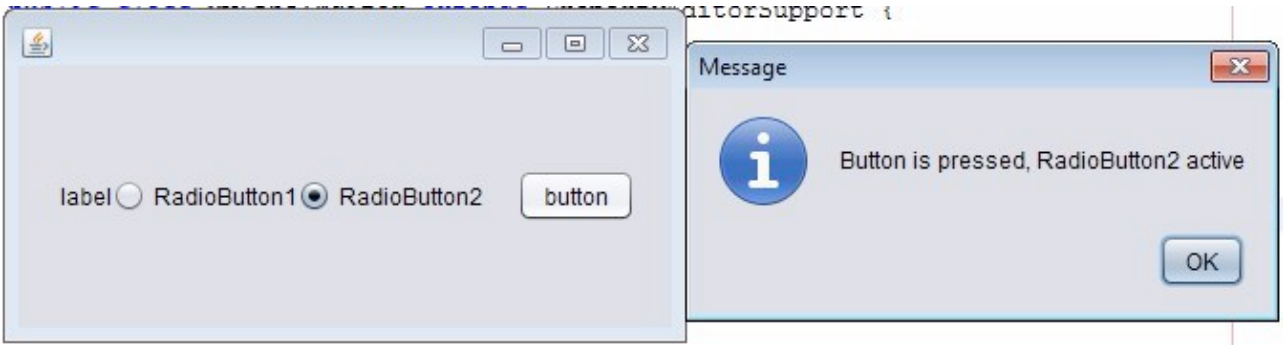

Рис. 9.1 – Использование компонента в *NetBeans*

9.3 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*.

10 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №10

Цель:

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

Предметная область:

Графический интерфейс пользователя.

10.1 Краткие теоретические сведения

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

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

10.2 Задание

Постановка задачи:

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

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

Решение задачи:

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

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

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

Результат:

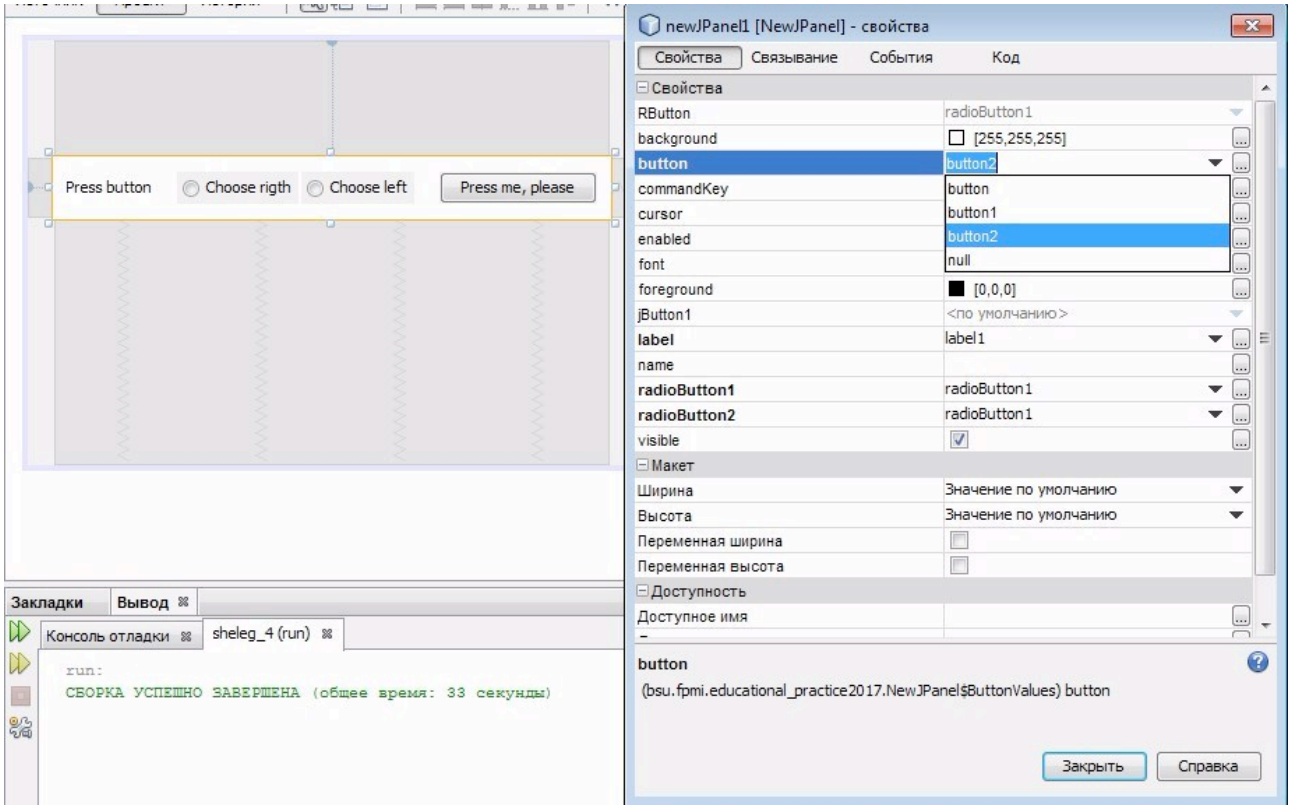

Рис. 10.1 – Выбор из допустимых значений для *button*

10.3 Выводы

- Настройку свойств собственных компонентов можно контролировать, если для каждого свойства создать свой класс, реализующий *PropertyEditor*. Он будет отвечать за редактор настроек привязанного к нему свойства.
- Для настройки своих свойств можно создать свой класс-наследник *Component*, в котором сделать удобный для редактирования GUI.

11 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №11

Цель:

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

Предметная область:

Удаленный вызов процедур. Сетевые приложения.

11.1 Краткие теоретические сведения

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

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

Для того чтобы создать приложение на базе RMI нужно:

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

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

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

4) При использовании RMI клиент и сервер не взаимодействуют непосредственно. На стороне клиента ссылка на удаленный объект реализуется в виде экземпляра класса заглушки. Когда клиент вызывает удаленный метод, в действительности вызывается метод объекта-заглушки. Заглушка производит необходимые сетевые операции по передаче этого вызова находящемуся на сервере классу каркаса. Этот каркас транслирует пришедший по сети запрос в вызов метода серверного объекта, а затем передает возвращенное им значение обратно заглушке, которая, в свою

очередь, возвращает его клиенту. Все это представляет собой довольно сложную систему, но прикладным программистам никогда не приходится думать о заглушках и каркасах; они генерируются автоматически утилитой *rmic*. До версии Java 5 заглушки приходилось создавать вручную, а в современных версиях Java 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.

11.2 Задание

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.

Учёт рабочего времени. Сервер ведёт учёт времени работы клиентов, данные сохраняются в файле. Клиент при запуске связывается с сервером и сообщает данные клиента. Сервер каждые 5 мин запрашивает подтверждение у клиента, что он ещё подключен. Если клиент не отвечает, он закончил работу.

Решение задачи:

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

Клиент реализован как консольное приложения. У каждого клиента запрашивается ввод имени и номера команды для дальнейшей работы.

Изначально запускается сервер, записывает в файл время начала работы. Далее, при каждом запуске клиента происходит установка соединения с сервером и в файл добавлется запись с именем клиента и временем его подключения к серверу. При удалении клиента или не активности клиента в течении 5 минут происходит дозапись об удалении клиента.

Результат:

Рис. 11.1 – Запуск сервера

Рис. 11.2 – Запуск клиента

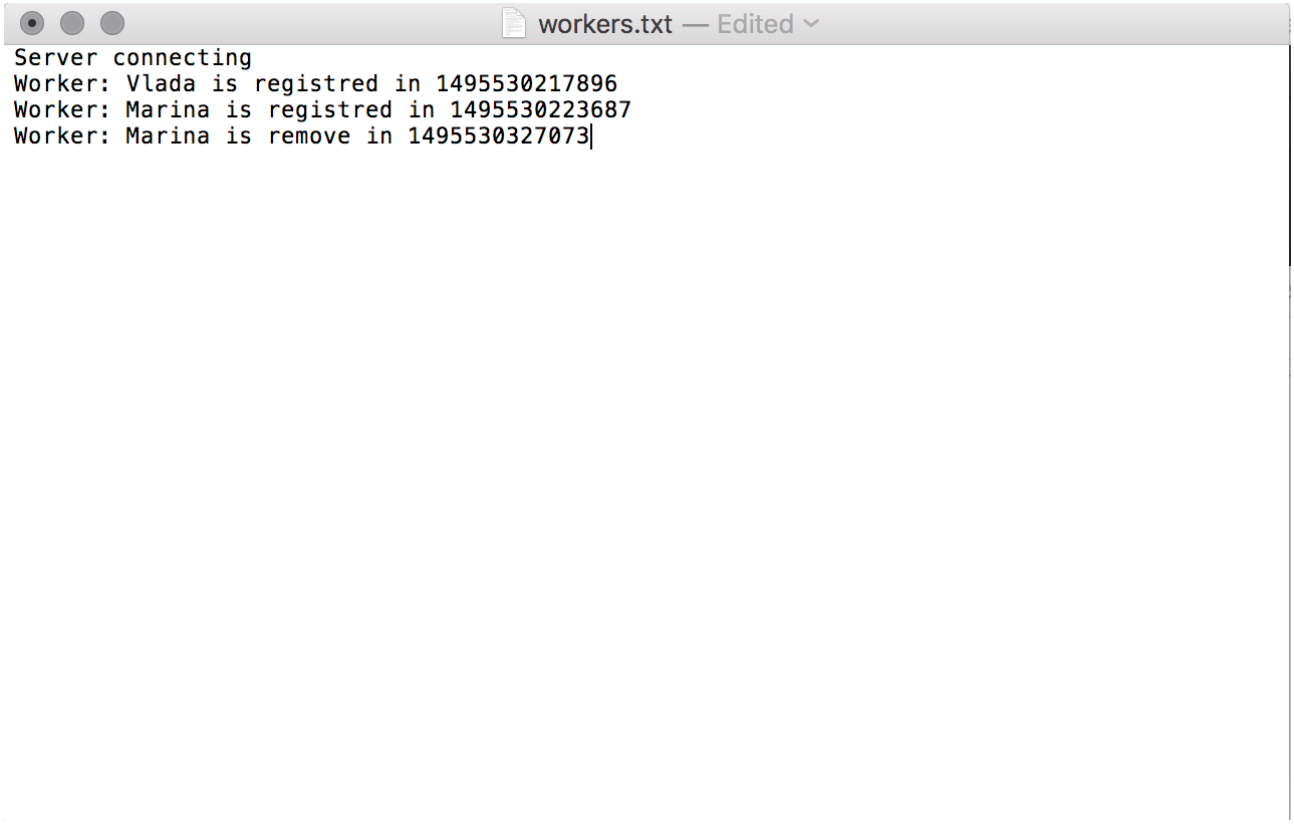

Рис. 11.3 – Файл с учетом данных работников

11.3 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*.

12 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №12

Цель:

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

Предметная область:

Сетевые приложения. WEB-разработка.

12.1 Краткие теоретические сведения

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

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

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

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

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

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

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

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

Контейнеры сервлетов создаются как часть Web-сервера или как встраиваемый в него модуль. Большую популярность получили встраиваемые контейнеры Tomcat, разработанные сообществом Apache Software Foundition (http://tomcat.apache.org/).

12.2 Задание

Постановка задачи:

Создать сервлет и взаимодействующие с ним пакеты Java-классов и HTML-документов, выполняющие действия для решения вашего варианта задания. Представить решение в виде web-приложения (как в примере). Необходимо было реализовать выбор текстового файла из архива файлов по разделам (поэзия, проза, фантастика и т.д.) и его отображение

Решение задачи:

Были скачаны установлены Apache Tomcat версии 8.0.22 и Apache Ant 1.9.4. После этого были рассмотрены примеры по адресу http://localhost:8080/examples/servlets/ и http://localhost:8080/examples/jsp/.

Результат:

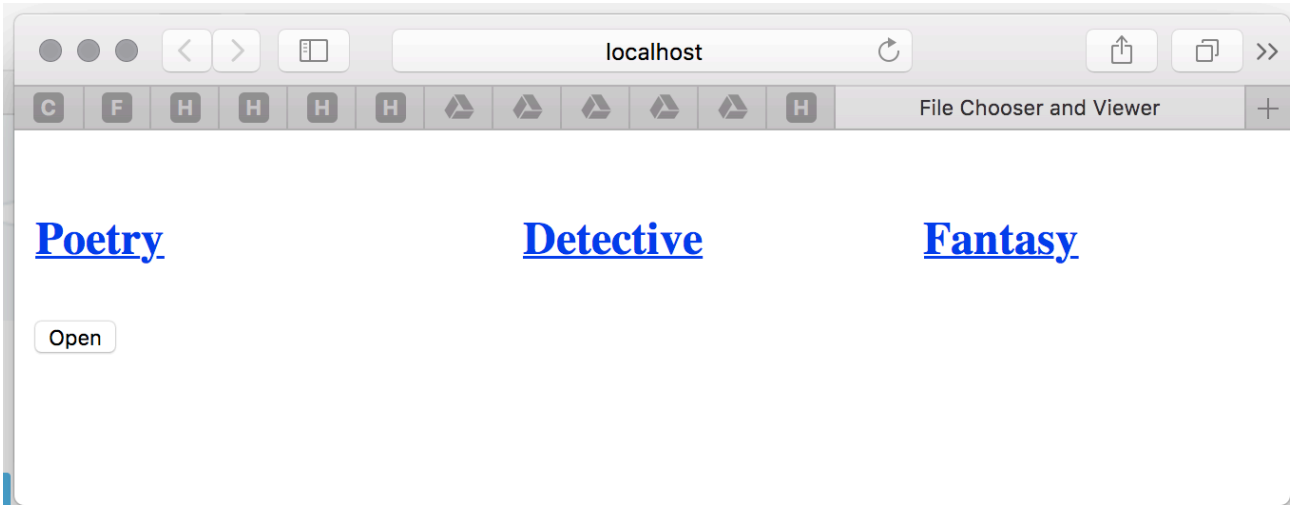

Рис. 12.1 – Скриншот главного окна

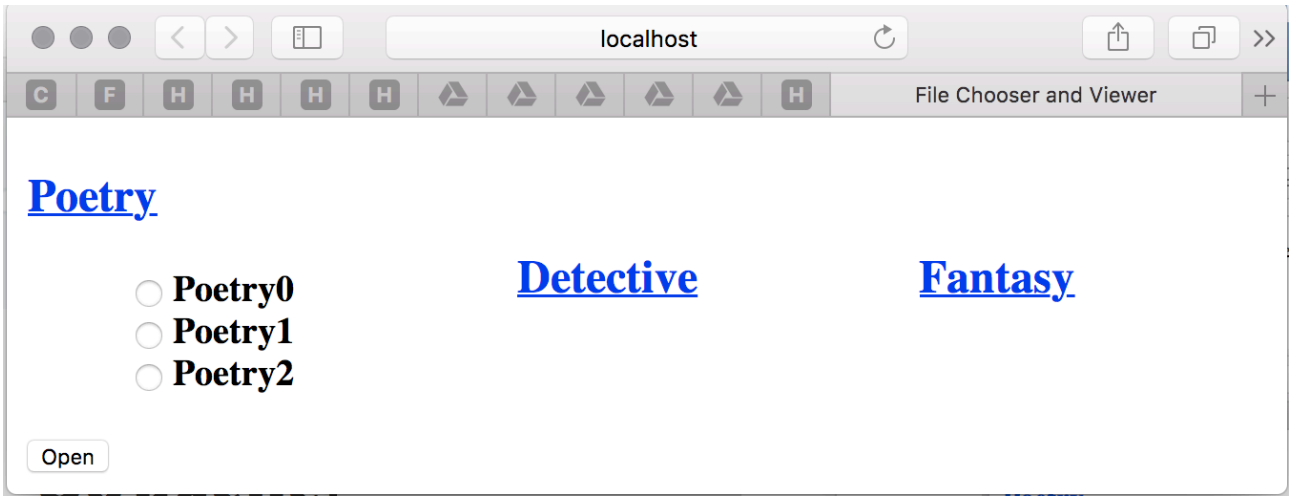

Рис. 12.2 – Скриншот главного окна при нажатии на одну из ссылок

12.3 Выводы

- В качестве веб-сервера для сервлетов можно использовать бесплатный Apache Tomcat.
- При помощи Java можно создавать сервлеты, при помощи которых можно создавать динамические веб-сайты.
- При разработке веб-приложений можно использовать как сервлеты, так и технологию JSP (JavaServer Pages).

13 ОТЧЕТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ №13

Цель:

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

Предметная область:

Разработка библиотек.

13.1 Краткие теоретические сведения

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

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

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

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

13.2 Задание

Постановка задачи:

- Разработайте класс Java для решения вашего варианта задания с помощью native методов.
- Создайте тестовое приложение (одно или несколько) для демонстрации всех возможностей вашей разработки
- Используйте MS Visual Studio C/C++ и Win32 API для разработки кода, зависящего от платформы, при необходимости можно использовать встроенный ассемблер.

Реализуйте работу с реестром Windows (см. функции Win32 API Reg*: RegOpenKeyEx, RegCloseKey, RegCreateKeyEx, RegDeleteKey, RegDeleteValue, RegQueryValueEx, RegSetValueEx, и т.п.).

Решение задачи:

Функции для работы с реестром хранятся в библиотеке *RegKey.dll*. В ней содержатся нативные методы *RegCreateKeyEx, RegOpenKeyEx, RegDeleteKeyEx, RegCloseKey, RegDeleteValue, RegSetValueEx, RegGetValue*.

Демонстрационное приложение реализовано в виде консоли.

Для проверки работы программы откроем реестр (Win+R, regedit) и удостоверимся, что запись добавлется.

Результат:

```
C:\Users\Vlada\Downloads\Public\Public>java Test

The subkey was successfully created.

The subkey was successfully opened.

The value of the key was set successfully.

The subkey was successfully closed.
```

Рис. 13.1 – Консоль после выполнения программы

Рис. 13.2 – Реестр после выполнения программы

13.3 Выводы

- При помощи технологии JNI можно запускать нативный код приложениях на Java.
- Перед тем, как использовать методы из библиотеки, необходимо ее предварительно загрузить, вызвав метод *System.loadLibrary*.
- При разработке библиотеки для JNI допускается вызов методов виртуальной машины через параметр типа *JNIEnv**.