Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

Факультет ИУ – «Информатика и управление»
Кафедра ИУ-3 – «Информационные системы и телекоммуникации»

Методические указания к лабораторным работам по курсу «Распределенные информационные системы»

Продолжительность работы: 4 часа

Составители: к.т.н. Локтев Д.А.,
Ершов В.А., Страхов М.Д.

2018

Содержание

Введение

Данное методическое пособие описывает работу по созданию клиент-серверного приложения на базе JavaFX. Курс состоит из трех лабораторных работ, в результате выполнения которых Вы:

– создадите приложение для локального сервера;

– создадите приложение для обмена данными между сервером Heroku и локальным сервером (Java, Groovy);

– создадите приложение JavaFX (Java, MySQL);

– интегрируете разработанные модули в единое клиент-серверное приложение.

Порядок выполнения лабораторных работ:

Глава 1. Архитектура.

1. Git. Учимся записывать и выгружать данные с репозитория.
2. Описание архитектуры проекта.
3. Локальное позиционирование Базы Данных MySQL.
4. JDBC. Будущая синхронизация баз данных

Глава 2. Серверная часть.

1. Установка и знакомство с Heroku.
2. Базовый проект Heroku.
3. POST/GET методы. Дорабатываем код.

Глава 3. Клиентская часть.

1. JavaFX. Контейнеры. FXML.
2. Простая программа с локальным сервером.
3. Синхронизируем с Heroku.
4. Кастомизация.

Для выполнения трех лабораторных работ Вам понадобится:

1. IntelliJ IDE (все примеры и структура проекта будут представлены именно в IntelliJ, однако Вы можете использовать Eclipse для выполнения лабораторных работ)
2. MySQL Server (ссылки и описание установки смотреть в методических указаниях в разделе работы с базой данных, а также в списке литературы).
3. Heroku (ссылки и описание установки смотреть в методических указаниях в разделе работы с базой данных, а также в списке литературы).
4. Java 1.8 (ссылки и описание установки смотреть в методических указаниях в разделе работы с базой данных, а также в списке литературы).

Лабораторная работа 1

Задание:

1. Создать базу данных, состоящую из двух таблиц:
   - таблица логинизации, в которой будут храниться данные для авторизации пользователя (login – primary key, password, user_name);
   - вторая таблица делается соответственно варианту (в таблице должно быть не менее 100 уникальных записей).
2. Создать проект на Java с соединением к базе данных (к обеим таблицам) и с реализацией следующих методов:
   - Создание таблицы – CREATE;
   - Удаление таблицы – DROP;
   - Вставки данных – INSERT;
   - Удаление данных – DELETE;
   - Выборка всех данных – SELECT * FROM table;
   - Выборка конкретных данных – SELECT * FROM table WHERE <condition>.
3. Создать тестовый класс на основе Junit 4, который протестирует функционал данного проекта в следующем порядке:
   1) создание таблицы,
   2) вставка данных в таблицу 1,
   3) вставка данных в таблицу 2,
   4) выборка всех данных (из таблицы 1 и/или 2),
   5) выборка определенных данных,
   6) удаление данных,
   7) удаление таблицы.
4. Залить файл готового проекта на Git репозиторий.

Важно! ***Не реализовывайте весь функционал в одном методе/классе!!!*** Пример грамотного кода можно посмотреть *здесь*.

В списке литературы Вы найдете пример программы, а также описание проблем, с которыми Вы можете столкнуться при выполнении данной лабораторной работы.

Глава 1. Архитектура.

Git. Учимся записывать и выгружать данные с репозитория

GIT – система версионного контроля, которая находится в свободном доступе. В данный момент, существует много разновидностей гита, которые позволяют разработчикам хранить данные в удаленном месте, т.н. облаке, изменять данные, не боясь в случае неправильных правок откатить изменения.

На данный момент самыми популярными версиями гита являются:

– Githhub

– Bitbucket

Для создания локального git репозитория понадобиться выполнить следующие действия:

1) Создать директорию проекта /path/to/my/codebase

```
$ cd /path/to/my/codebase
```

2) Инициализировать локальный git-репозиторий и создать каталог /.git

```
$ git init
```

Добавление данных в репозиторий происходит в несколько этапов:

1) Добавить все файлы в индексацию (в дальнейшем будет добавлять только те, которые были изменены).

```
$ git add .
```

2) Добавить комментарий к новой точке индексации файлов

```
$ git commit -m "comment"
```

3) Положить измененные данные в origin, в ветку master

```
$ git push origin master
```

На самом деле веток может быть несколько. Это дает Вам возможность работать с кодом, не изменяя главную ветку реализации продукта. То есть это

можно рассматривать как возможность работы с модифицированной копией проекта.

Чтобы создать новую ветку используются следующие команды:

```
$ git branch new_branch_name
```

После того, как Вы ввели данную команду, git автоматически создаст новую ветку под названием new_branch_name и сделает ее вашей основной веткой. То есть, если вы захотите залить изменения (git push), то они будут сделаны именно на ветку new_branch_name.

Чтобы перейти на другую ветку, необходимо ввести команду:

```
$ git checkout branch_name
```

После этого вы будете перенесены на ветку с именем branch_name.

Вместо этих двух команд можно использовать идентичную более сокращенную команду:

```
$ git checkout -b branch_name
```

Для того, чтобы выгрузить данные с гита (допустим, вы перешли на другой компьютер, у вас есть установленный для работы софт, но нет проекта, который лежит в определенной директории, скажем, github-a), необходимо ввести команду:

```
$ git pull origin master
```

Эта команда выгрузит данные с ветки master (последние изменения кода).

Если вы работаете с удаленным репозиторием, то Вам понадобятся также выполнить команды git clone и git remote. Подробнее с настройкой git репозиториев можно ознакомится здесь .

Описание архитектуры проекта

Данный проект будет содержать 3 части, которые будут связаны между собой некоторыми интерфейсами:

1. Локальная база данных и код, который бы генерировал или записывал данные в базу данных (для инкапсуляции и защиты доступа к базе данных)
2. Серверная часть, которая будет позволять пользователю обращаться на удаленную базу данных, выгружать и загружать данные для их дальнейшей обработки и использования.
3. Клиентское приложение, которое Вам непосредственно будет предложено реализовать соответственно с вариантом задания.

Таким образом, всю систему можно представить следующим образом (рис.1):

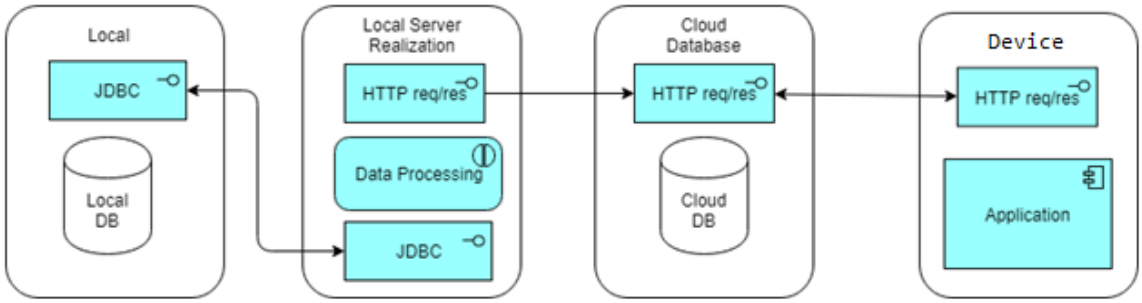

Рисунок 1. Общая структура создаваемого приложения.

Из рис.1 видно, что приложение будет напрямую общаться с базой данных, которая будет реализована как облачное хранилище. Данное облачное хранилище будет заполняться с помощью локальной программы, которая позволит нам эмитировать синхронизацию баз данных.

Локальная база данных может быть реализована с помощью любой СУБД. Однако в данной работе для реализации проекта будет использоваться MySQL, предоставляемая корпорацией Microsoft для локального развертывания.

Для установки соединения с базой данных, заполнения ее данными, выгрузки данных, их обработки и отправление на облако будет использоваться язык Java (а также реализация JDBC, также предоставляемая корпорацией Microsoft для взаимодействия с базами данных).

Для реализации приложения будет также использоваться язык программирования Java с использованием актуальных библиотек по реализации приложений (JavaFX).

Таким образом, в конце данных лабораторных работ Вы получите приложение, которое будет иметь возможность работать с данными, положенными в удаленную базу данных, а также приложение (которое можно будет в дальнейшем оснастить графической реализацией) для обработки данных локально.

Локальное позиционирование Базы Данных MySQL

Установка MySQL сервера описана на официальном сайте MySQL: https://dev.mysql.com/downloads/installer/

Внимание! Есть пункты, на которые требуется обратить внимание:

1. Не настраивайте сложную маршрутизацию через роутер или синхронизацию БД.
2. Не настраивайте SSL. Это лишь усложнит вашу задачу.
3. Рекомендуется отдельно скачать JDBC (Java DataBase Connector) https://dev.mysql.com/downloads/connector/j/

Когда вы установите все необходимое ПО, на вашем компьютере должны появится 3 главные вещи:

1. MySQL Workbench – предоставляемая IDE для работы с SQL.
2. JDBC
3. Служба запуска сервера MySQL. Ее можно найти в разделе “Службы” (поиск по компьютеру). (автоматическое подключение).

Если Вы сделали все правильно, то в разделе Server Status (после входа в MySQL Connections) отобразится следующее:

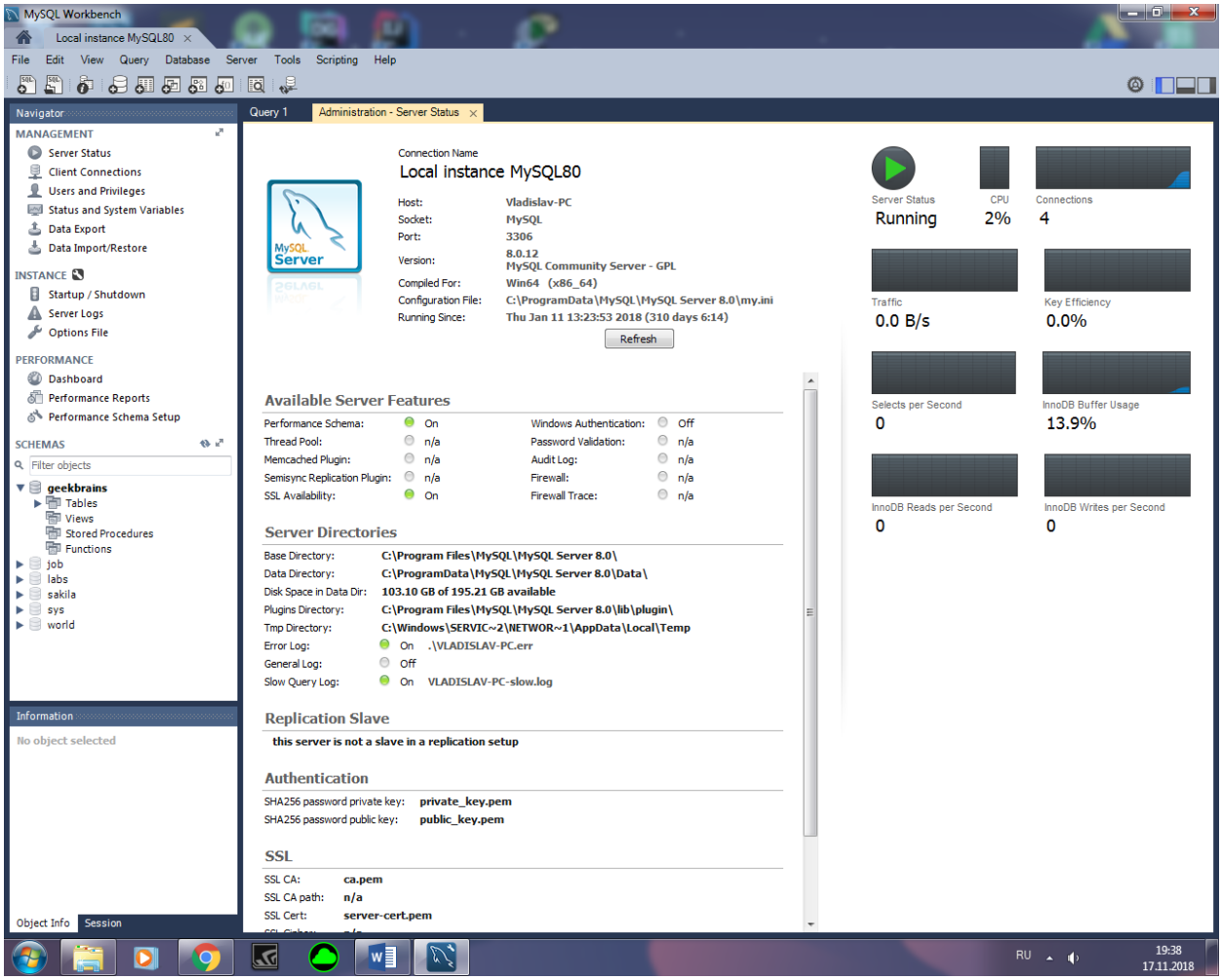

Рисунок 2. Правильная конфигурация показывает, что MySQL Server работает.

Теперь Вы можете приступить к созданию таблиц.

JDBC. Будущая синхронизация баз данных

JDBC (Java DataBase Connectivity — соединение с базами данных на Java) — платформенно-независимый промышленный стандарт взаимодействия Java-приложений с различными СУБД, реализованный в виде пакета java.sql, входящего в состав Java SE.

JDBC основан на концепции так называемых драйверов, позволяющих получать соединение с базой данных по специально описанному URL.

Интерфейсы

Соединение с базой данных описывается классом, реализующим интерфейс **java.sql.Connection**. Имея соединение с базой данных, можно создавать объекты типа **Statement**, служащие для исполнения запросов к базе данных на языке SQL.

Существуют следующие виды типов Statement, различающихся по назначению:

– java.sql.Statement — Statement общего назначения;

– java.sql.PreparedStatement — Statement, служащий для выполнения запросов, содержащих подставляемые параметры (обозначаются символом '?' в теле запроса);

– java.sql.CallableStatement — Statement, предназначенный для вызова хранимых процедур.

– Интерфейс java.sql.ResultSet позволяет легко обрабатывать результаты запроса.

Основные команды.

Для работы с базой данных необходимо определить параметры авторизации пользователя:

```
final String DATABASE_URL = "jdbc:mysql://localhost/database_name";
final String USER = "root";
final String PASSWORD = "1234";
```

Данные параметры можно передавать с применением командной строки или через консоль IDE, чтобы защитить базу данных от постороннего вмешательства, создав иерархию пользователей, однако в данной лабораторной работе ограничимся только использованием констант.

Данные URL, USER, PASSWORD Вы можете узнать, непосредственно открыв настройки вашей базы данных. Так, если Вы используете MySQL Workbench от Microsoft, то данные входа будут отображены во вкладке Administration – Users and Privileges (рис. 3).

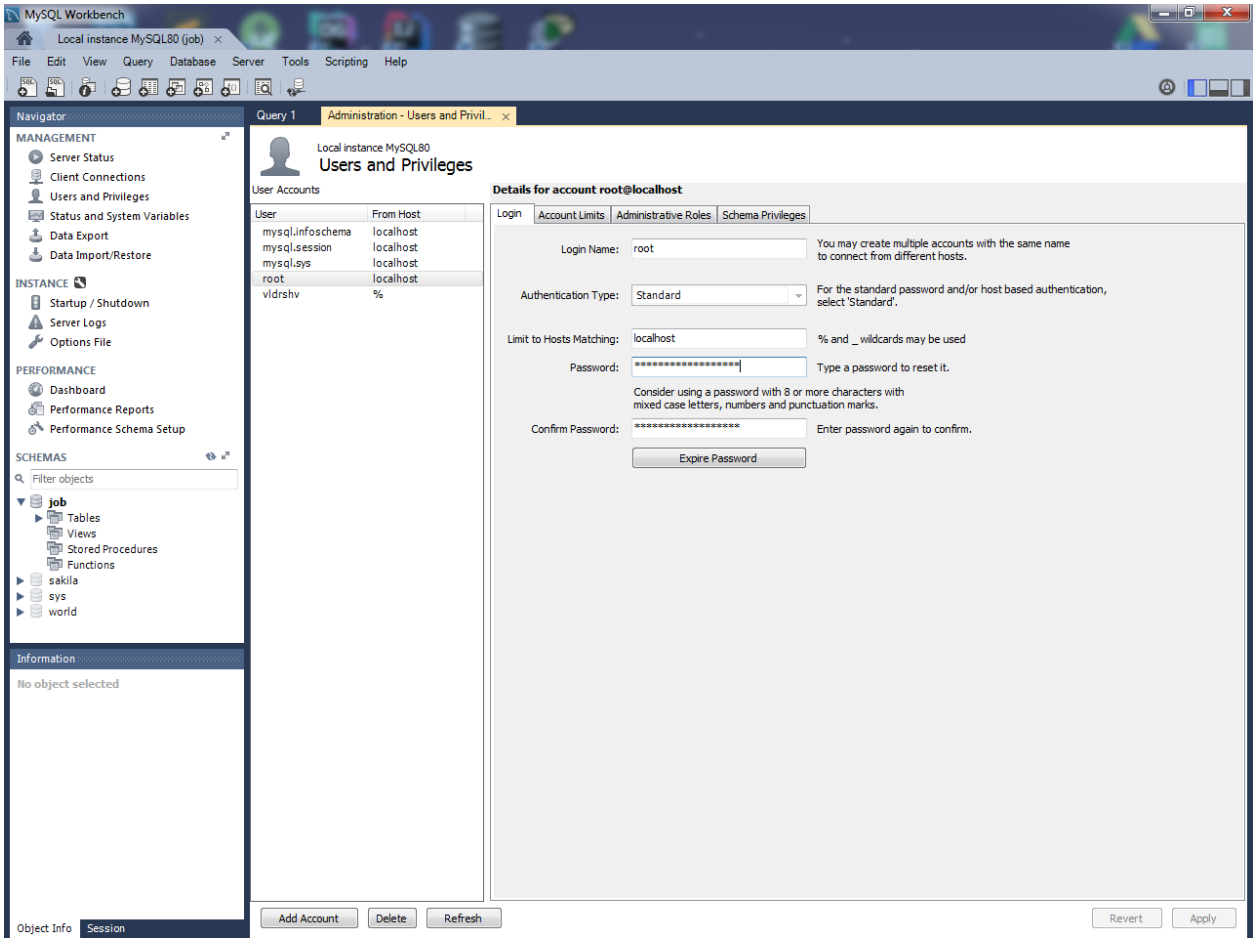

Рисунок 3. Отображение данных входа в БД.

Важно! Создавайте простой пароль для БД, которой могли бы легко запомнить, т.к. он понадобиться Вам в следующих этапах. В данный момент создание сложного пароля для сбережения вашей базы данных от взлома не является приоритетной задачей. Также обратите внимание, что используется пароль в зашифрованном виде!

Для работы с базой данных соблюдается следующий алгоритм:

1. Логинизация или аутентификация. Вы должны подключиться к БД с помощью URL, USER и PASSWORD.
2. Составление запроса к базе данных.
3. Применение запроса к базам данных.
4. Получение ответа от базы данных.
5. Формирования объекта по полученному ответу.
6. Работа с объектом.

Данная работа проиллюстрирована в листинге 1 в конце данного пособия.

В листинге имеется 3 важных объекта:

1. Connection con – используется для создания связи с базой данных и получения данных от БД.
2. Statement stmt – используется для отправки запроса в базу данных. С помощью данного объекта Вы можете взаимодействовать с базой данных и получать ровно те данные, о которых запросите. В объект передается сформированная строка SQL-запроса, которая будет интерпретирована самой базой данных.
3. ResultSet rs – используется как контейнер полученных результатов. Важно использовать структуру while, которая описана в данном листинге. Благодаря ей Вы проходите по всем значениям полученного результата (а именно множеству) и выгружаете данные из таблицы. Важно понимать, что выгрузка происходит благодаря следующим методам:
    a. getInt(column_index);
    b. getString(column_index);

c. getShort(column_index);

d. getLong(column_index);

e. и так далее.

Также необходимо понимать, что работа с базой данных – важнейший ресурс, на который JVM выделяет память и ресурсы процессора, занимая его время обработкой данных, а также поддержкой подключения. Поэтому в конце программы необходимо закрыть те интерфейсы, которые Вы использовали для работы с базой данных.

Если Ваша таблица не была создана, нет вообще такой базы данных или неправильные данные аутентификации – будет выброшено исключение типа SQLException, которое необходимо перехватывать и обрабатывать. В противном случае выводить данные об ошибке.

Список литературы (глава 1).

1. https://tproger.ru/translations/java-jdbc-example/
2. https://proselyte.net/tutorials/jdbc/simple-application-example/

Лабораторная работа 2

Задание

1) Разобраться с работой сервера;
2) Создать программу, которая работать будет как локальный сервер;
3) Ознакомится с реализацией клиентской части.
4) Скачать Heroku CLI. Запустить Hello-world-template;
5) На основании сделанного проекта реализовать свой проект.

Глава 2. Серверная часть.

Основы работы в Сети

В основу работы в сети, поддерживаемой в Java, положено понятие сокета, обозначающего конечную точку в Сети. Сокеты составляют основу современных способов работы в Сети, поскольку сокет позволяет отдельному компьютеру одновременно обслуживать много разных клиентов, предоставляя разные виды информации. Эта цель достигается благодаря применению порта — нумерованного сокета на отдельной машине.

Говорят, что серверный процесс «прослушивает» порт до тех пор, пока клиент не соединится с ним. Сервер в состоянии принять запросы от многих клиентов, подключаемых к порту с одним и тем же номером, хотя каждый сеанс связи индивидуален. Для управления соединениями со многими клиентами серверный процесс должен быть многопоточным или располагать какими-то другими средствами для мультиплексирования одновременного ввода-вывода.

Связь между сокетами устанавливается и поддерживается по определённому сетевому протоколу. Протокол Интернета (IP) является низкоуровневым маршрутизирующим сетевым протоколом, разбивающим данные на небольшие пакеты и посылающим их через Сеть по определенному адресу, что не гарантирует доставки всех этих пакетов по этому адресу. Протокол управления передачи (TCP) является сетевым протоколом более высокого уровня, обеспечивающим связывание, сортировку и повторную передачу пакетов, чтобы обеспечить надежную доставку данных. Ещё одним сетевым протоколом более низкого уровня, чем TCP, является протокол пользовательских дейтаграмм (UDP). Этот сетевой протокол может быть использован непосредственно для поддержки быстрой, не требующей постоянного соединения и ненадежной транспортировки пакетов.

Как только соединение будет установлено, в действие вступает высокоуровневый протокол, тип которого зависит от используемого порта. Протокол TCP/IP резервирует первые 1 024 порта для отдельных протоколов. Например, порт 21 выделен для протокола FTP, порт 23 — для протокола Telnet, порт 25 — для электронной почты, порт 80 — для протокола HTTP и т.д. Каждый сетевой протокол определяет порядок взаимодействия клиента с портом.

Например, протокол HTTP используется серверами и веб-браузерами для передачи гипертекста и графических изображений. Это довольно простой протокол для базового постраничного просмотра информации, предоставляемой веб-серверами. Рассмотрим принцип его действия. Когда клиент запрашивает файл у HTTP-сервера, это действие называется обращением. Оно состоит в том, чтобы отправить имя файла в специальном формате в предопределённый порт и затем прочитать содержимое этого файла. Сервер также сообщает код состояния, чтобы известить клиента, был ли запрос обслужен, а также причину, по которой он не может быть обслужен.

Главной составляющей Интернета является адрес, который есть у каждого компьютера в Сети. Изначально все адреса состояли из 32-разрядных значений, организованных по четыре 8-разрядных значения. Адрес такого типа определён в протоколе IPv4. Но в последнее время вступила в действие новая схема адресации, называемая IPv6 и предназначенная для поддержки намного большего адресного пространства. Правда, для сетевого программирования на Java обычно не приходится беспокоиться, какого типа адрес используется: IPv4 или IPv6, поскольку эта задача решается в Java автоматически.

Сокеты по протоколу TCP/IP служат для реализации надежных двунаправленных постоянных двухточечных потоковых соединений между хостами в Интернете. Сокет может служить для подключения системы ввода-вывода в Java к другим программам, которые могут находиться как на локальной машине, так и на любой другой в Интернете.

В Java поддерживаются две разновидности сокетов по протоколу TCP/IP: один — для серверов, другой — для клиентов. Класс ServerSocket предназначен для создания сервера, который будет обрабатывать клиентские подключения, тогда как класс Socket предназначен для обмена данными между сервером и клиентами по сетевому протоколу. При создании объекта типа Socket неявно устанавливается соединение клиента с сервером.

Для доступа к потокам ввода-вывода, связанным с классом Socket, можно воспользоваться методами getInputStream() и getOuptutStream(). Каждый из этих методов может сгенерировать исключение типа IOException, если сокет оказался недействительным из-за потери соединения. Эти потоки ввода-вывода используются для передачи и приема данных.

Написание эхо-сервера

```
public class MainClass {
    public static void main(String[] args) {
        ServerSocket serv = null;
        Socket sock = null;
        try {
            serv = new ServerSocket(8189);
            System.out.println("Сервер запущен, ожидаем подключения...");
            sock = serv.accept();
            System.out.println("Клиент подключился");
            Scanner sc = new Scanner(sock.getInputStream());
            PrintWriter pw = new PrintWriter(sock.getOutputStream());
            while (true) {
                String str = sc.nextLine();
                if (str.equals("end")) break;
                pw.println("Эхо: " + str);
                pw.flush();
            }
        } catch (IOException e) {
            System.out.println("Ошибка инициализации сервера");
        } finally {
            try {
                serv.close();
```

```
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Для начала создаётся объект класса ServerSocket, представляющий собой сервер, который прослушивает порт 8189. Метод server.accept() переводит основной поток в режим ожидания, поэтому, пока никто не подключится, следующая строка кода выполнена не будет. Как только клиент подключился, информация о соединении с ним запишется в объект типа Socket. Для обмена сообщениями с клиентом необходимо создать обработчики входящего и исходящего потока, в данном случае это — Scanner и PrintWriter, в дальнейшем будем использовать DataInputStream и DataOutputStream.

Поскольку мы создаём эхо-сервер, обработка данных производится следующим образом: сервер считывает сообщение, переданное клиентом, добавляет к нему фразу «Эхо: » и отсылает обратно. Если клиент прислал сообщение «end», общение с ним прекращается, и сокет закрывается.

Блок finally предназначен для гарантированного закрытия всех сетевых соединений и освобождения ресурсов.

Написание клиентской части

Ниже представлен код клиентской части чата.

```
public class MainClass {
    public static void main(String[] args) {
        new MyWindow();
    }
}
public class MyWindow extends JFrame {
    private JTextField jtf;
    private JTextArea jta;
    private final String SERVER_ADDR = "localhost";
    private final int SERVER_PORT = 8189;
```

```
    private Socket sock;
    private Scanner in;
    private PrintWriter out;
    public MyWindow() {
        try {
            sock = new Socket(SERVER_ADDR, SERVER_PORT);
            in = new Scanner(sock.getInputStream());
            out = new PrintWriter(sock.getOutputStream());
        } catch (IOException e) {
            e.printStackTrace();
        }
        setBounds(600, 300, 500, 500);
        setTitle("Client");
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        jta = new JTextArea();
        jta.setEditable(false);
        jta.setLineWrap(true);
        JScrollPane jsp = new JScrollPane(jta);
        add(jsp, BorderLayout.CENTER);
        JPanel bottomPanel = new JPanel(new BorderLayout());
        add(bottomPanel, BorderLayout.SOUTH);
        JButton jbSend = new JButton("SEND");
        bottomPanel.add(jbSend, BorderLayout.EAST);
        jtf = new JTextField();
        bottomPanel.add(jtf, BorderLayout.CENTER);
              jbSend.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (!jtf.getText().trim().isEmpty()) {
                    sendMsg();
                    jtf.grabFocus();
                }
            }
        });
        jtf.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                sendMsg();
            }
        });
        new Thread(new Runnable() {
```

```
            @Override
            public void run() {
                try {
                    while (true) {
                        if (in.hasNext()) {
            String w = in.nextLine();
                            if (w.equalsIgnoreCase("end session")) break;
                            jta.append(w);
                            jta.append("\n");
                        }
                    }
                } catch (Exception e) {
                }
            }
        }).start();
        addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                super.windowClosing(e);
                try {
                    out.println("end");
                    out.flush();
                    sock.close();
                    out.close();
                    in.close();
                } catch (IOException exc) {
                }
            }
        });
        setVisible(true);
    }
    public void sendMsg() {
        out.println(jtf.getText());
        out.flush();
        jtf.setText("");
    }
}
```

Большая часть приведённого выше кода связана с созданием графического интерфейса. Поэтому разберём отдельные блоки, не касающиеся GUI.

```
public class MyWindow extends JFrame {
    // ...

          private final String SERVER_ADDR = "localhost";
    private final int SERVER_PORT = 8189;
    private Socket sock;
    private Scanner in;
    private PrintWriter out;
    public MyWindow() {
        try {
            sock = new Socket(SERVER_ADDR, SERVER_PORT);
            in = new Scanner(sock.getInputStream());
            out = new PrintWriter(sock.getOutputStream());
        } catch (IOException e) {
            e.printStackTrace();
        }
         // ...
         }
      }
```

Константа SERVER_ADDR задаёт адрес сервера, к которому будет подключаться клиент, SERVER_PORT — номер порта. Для открытия соединения с сервером и обмена сообщениями используются объекты классов Socket, Scanner и PrintWriter, по аналогии с серверной частью.

```
new Thread(new Runnable() {
    @Override
    public void run() {
        try {
            while (true) {
                if (in.hasNext()) {
                    String w = in.nextLine();
                    if (w.equalsIgnoreCase("end session")) break;
                    jta.append(w);
                    jta.append("\n");
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
```

```
    }
}).start();
```

Чтение данных с сервера организовано в отдельном потоке, как показано выше. Производится постоянный опрос входящего потока на наличие данных, если какие-то данные пришли, мы их записываем в многострочное текстовое поле на форме с помощью метода append(). Отдельный поток для чтения сделан для того, чтобы бесконечный цикл полностью не блокировал работу программы.

```
public void sendMsg() {
    out.println(jtf.getText());
    out.flush();
    jtf.setText("");
}
```

Отсылка сообщений на сервер организована с помощью метода sendMsg(), который извлекает сообщение из однострочного текстового поля и пишет его в исходящий поток.

Знакомство с Heroku. Установка.

Heroku — облачная PaaS-платформа, поддерживающая ряд языков программирования. С 2010 года является дочерней компанией Salesforce.com. Heroku, одна из первых облачных платформ, появилась в июне 2007 года и изначально поддерживала только язык программирования Ruby, но на данный момент список поддерживаемых языков также включает в себя Java, Node.js, Scala, Clojure, Python, Go и PHP. На серверах Heroku используются операционные системы Debian или Ubuntu.

Установка Heroku.

Для установки Heroku Вам понадобится:

1. Maven/Gradle (настоятельно рекомендуется использовать Gradle, т.к. он на данный момент наиболее популярный и проще в использовании)
2. Git
3. Java 1.8+
4. Знания английского и прямые руки.

Описание установки доступно на сайте (предварительно нужно зарегистрироваться на бесплатную версию):

https://devcenter.heroku.com/articles/getting-started-with-gradle-on-heroku

После установки у вас должен быть готовый проект Heroku (Hello World).

В ссылке к этому уроку прикреплен проект Heroku. Необходимо разобраться с кодом (Вы увидите, что половину кода Вы уже написали ранее и единственное, что Вам нужно сейчас подправить – MySQL диалект на SQLLite диалект и сделать выгрузку данный по GET-запросу).

Главное, в чем надо разобраться указано на рисунке ниже:

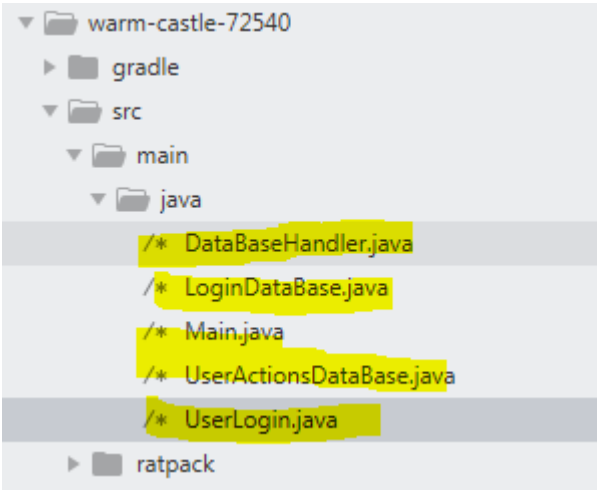

Рисунок 4. Часть клонированного проекта из списка полезных ссылок.

Сделать нужно по аналогии в соответствии с Вашим вариантом.

Список литературы (глава 2).

– Кей С. Хорстманн, Гари Корнелл. Java. Библиотека профессионала. Том 1. Основы // Пер. с англ. — М.: Вильямс, 2014. — 864 с.

– Брюс Эккель. Философия Java // 4-е изд.: Пер. с англ. — СПб.: Питер, 2016. — 1 168 с.

– Г. Шилдт. Java 8. Полное руководство // 9-е изд.: Пер. с англ. — М.: Вильямс, 2015. — 1 376 с.

– Г. Шилдт. Java 8: Руководство для начинающих. // 6-е изд.: Пер. с англ. — М.: Вильямс, 2015. — 720 с.

Полезные ссылки.

https://habr.com/post/232679/ - пример развертывания на Ruby

https://git.heroku.com/warm-castle-72540.git - пример проекта на Java с ответом в форме JSON.

Лабораторная работа 3

Задание:

1) Реализовать приложение, которое будет подключаться к базе данных и проверять наличие пользователя в этой базе данных. Если пользователя нет, то регистрировать его. Пользователей с одним user_name и почтой в базе данных быть не должно. Предусмотреть подсказку пользователю, что введенные им данные неверны.

   В приложении должны быть обязательно следующие поля:

   – поле ввода email, password;

   – кнопка log-in, log-out (можно одну кнопку, но текст на ней должен меняться в зависимости от того, вошел ли пользователь);

   – поле вывода информации из БД для соответствующего пользователя;

   – поле ввода данных (для последующей отправки в БД).

Примеры даны для библиотеки Swing. Необходимо разобраться в данном коде и портировать его на JavaFX (различие не большое, пример на github).

2) Реализовать подключение к Heroky (GET запрос отправить на сайт, в лабораторной работе 2 в примере проекта есть такой запрос к определенной странице, куда выгружаются данные в формате JSON).
3) Сохранить полученные данные в локальной таблице. Отобразить их в сделанном приложении.

Глава 3. Клиентская часть.

Создание формы

Для создания простого окна достаточно создать класс, унаследовать его от JFrame и создать объект этого класса в методе main(), как показано ниже:

```
public class MyWindow extends JFrame {
    public MyWindow() {
        setTitle("Test Window");
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        setBounds(300, 300, 400, 400);
        setVisible(true);
    }
}
      public class MainClass {
    public static void main(String[] args) {
        new MyWindow();
    }
}
```

В конструкторе сразу же задаются параметры окна:
setTitle() — устанавливает заголовок;
setDefaultCloseOperation(WindowConstants.*EXIT_ON_CLOSE*) — сообщает системе о необходимости завершить работу программы при закрытии окна;
setBounds() — задаёт координаты и размер формы в пикселях;
setVisible(true) — показывает форму на экране, желательно вызывать этот метод после всех настроек, иначе при запуске некоторые элементы могут быть отображены некорректно.

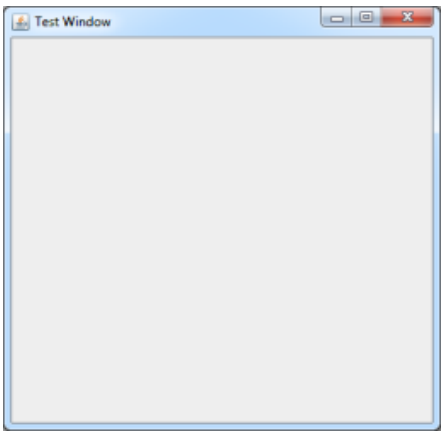

Рисунок 5. Пустое окно Swing

Попробуем добавить на форму несколько управляющих элементов, например, 5 кнопок JButton. Как правило, в библиотеке Swing названия классов, отвечающих за элементы графического интерфейса, начинаются с буквы J.

```
public class MyWindow extends JFrame {
    public MyWindow() {
        setTitle("Test Window");
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        setBounds(300, 300, 400, 400);
        JButton[] jbs = new JButton[5];
        for (int i = 0; i < 5; i++) {
            jbs[i] = new JButton("#" + i);
        }
        setLayout(new BorderLayout());    // выбор компоновщика элементов
        add(jbs[0], BorderLayout.EAST);  // добавление кнопки на форму
        add(jbs[1], BorderLayout.WEST);
        add(jbs[2], BorderLayout.SOUTH);
        add(jbs[3], BorderLayout.NORTH);
        add(jbs[4], BorderLayout.CENTER);
        setVisible(true);
    }
}
```

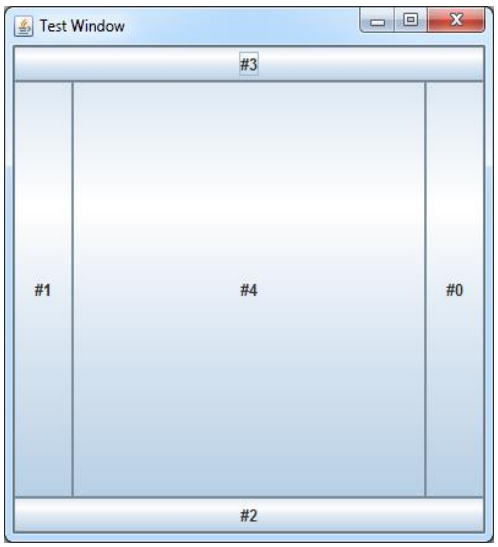

Рисунок 6. Форма с несколькими кнопками.

За расстановку элементов на форме отвечают компоновщики элементов, в данном случае мы использовали BorderLayout. После создания кнопок их необходимо добавить/расположить на форме, для этого используется метод add элемент_интерфейса, местонахождение.

Наиболее используемые компоновщики элементов

BorderLayout

BorderLayout — располагает элементы «по сторонам света» (запад, восток, север, юг и центр). Элемент, имеющий расположение CENTER, занимает бóльшую часть окна, то есть при растяжении формы сторонние элементы не будут менять размер, а центральный будет растягиваться, чтобы занять всю имеющуюся площадь.

```
public class MyWindow extends JFrame {
    public MyWindow() {
        setTitle("Test Window");
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        setBounds(300, 300, 400, 400);
        JButton button = new JButton("Button 1 (PAGE_START)");
        add(button, BorderLayout.PAGE_START);
        button = new JButton("Button 2 (CENTER)");
        button.setPreferredSize(new Dimension(200, 100));
        add(button, BorderLayout.CENTER);
        button = new JButton("Button 3 (LINE_START)");
        add(button, BorderLayout.LINE_START);
        button = new JButton("Long-Named Button 4 (PAGE_END)");
        add(button, BorderLayout.PAGE_END);
        button = new JButton("5 (LINE_END)");
        add(button, BorderLayout.LINE_END);
        setVisible(true);
    }
}
```

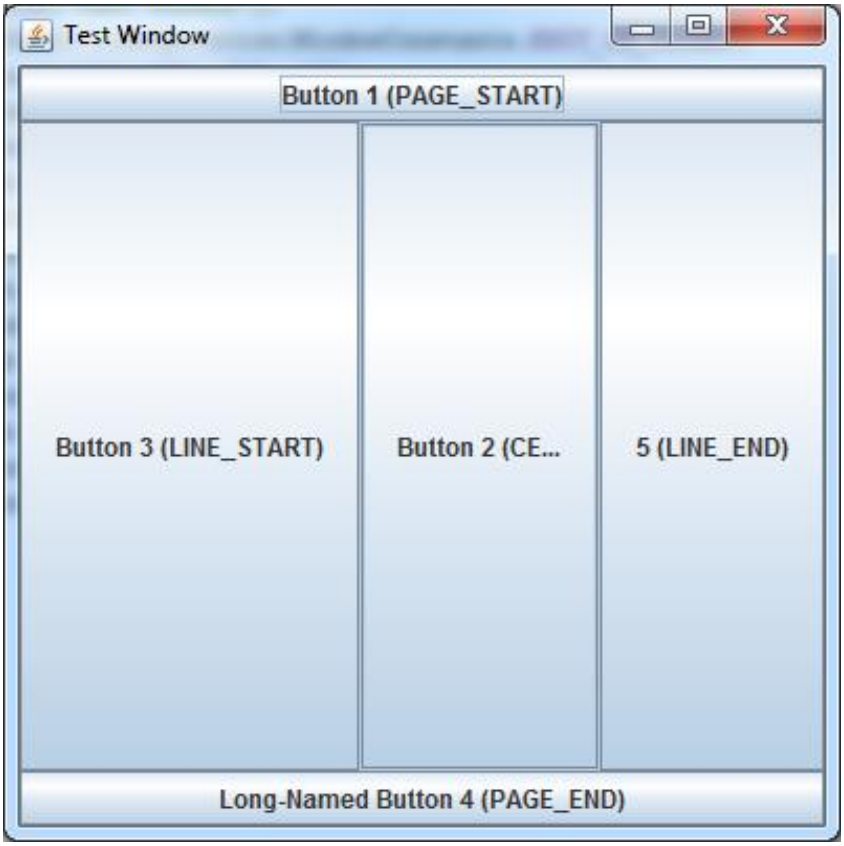

Рисунок 7. BorderLayout.

BoxLayout

BoxLayout — располагает элементы в строку или столбец, в зависимости от используемой константы: BoxLayout.Y_AXIS для расположения элементов в столбец, BoxLayout.X_AXIS — в строку.

```
public class MyWindow extends JFrame {
    public MyWindow() {
        setBounds(500,500,500,300);
        setTitle("BoxLayoutDemo");
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        JButton[] jbs = new JButton[10];
        setLayout(new BoxLayout(getContentPane(), BoxLayout.Y_AXIS)); // одну из
строк надо закомментировать
        setLayout(new BoxLayout(getContentPane(), BoxLayout.X_AXIS)); // одну из
строк надо закомментировать
        for (int i = 0; i < jbs.length; i++) {
            jbs[i] = new JButton("#" + i);
            jbs[i].setAlignmentX(CENTER_ALIGNMENT);
            add(jbs[i]);
        }
        setVisible(true);
    }
}
```

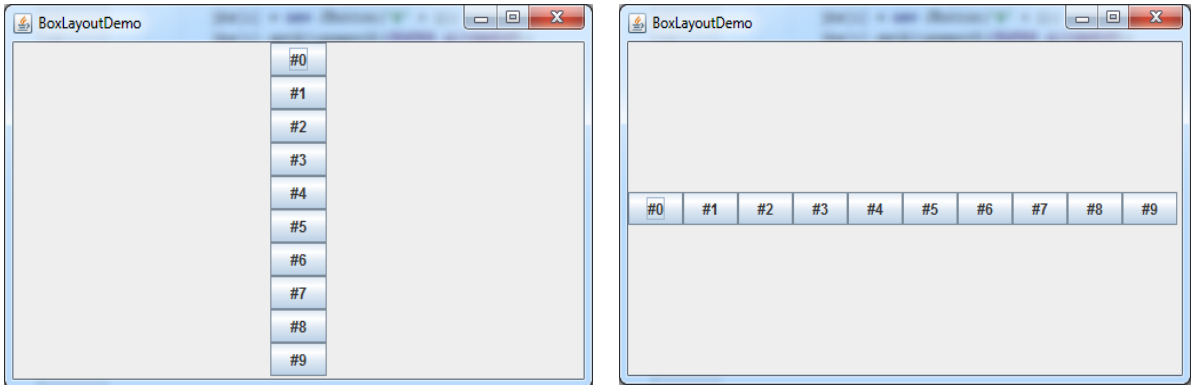

Рисунок 8. BoxLayout.

FlowLayout

FlowLayout — располагает элементы в одну строку, когда ширины строки становится недостаточно, переносит новые элементы на следующую.

```
public class MyWindow extends JFrame {
    public MyWindow() {
        setBounds(500, 500, 400, 300);
        setTitle("FlowLayoutDemo");
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        JButton[] jbs = new JButton[10];
        setLayout(new FlowLayout());
        for (int i = 0; i < jbs.length; i++) {
            jbs[i] = new JButton("#" + i);
            add(jbs[i]);
        }
        setVisible(true);
    }
}
```

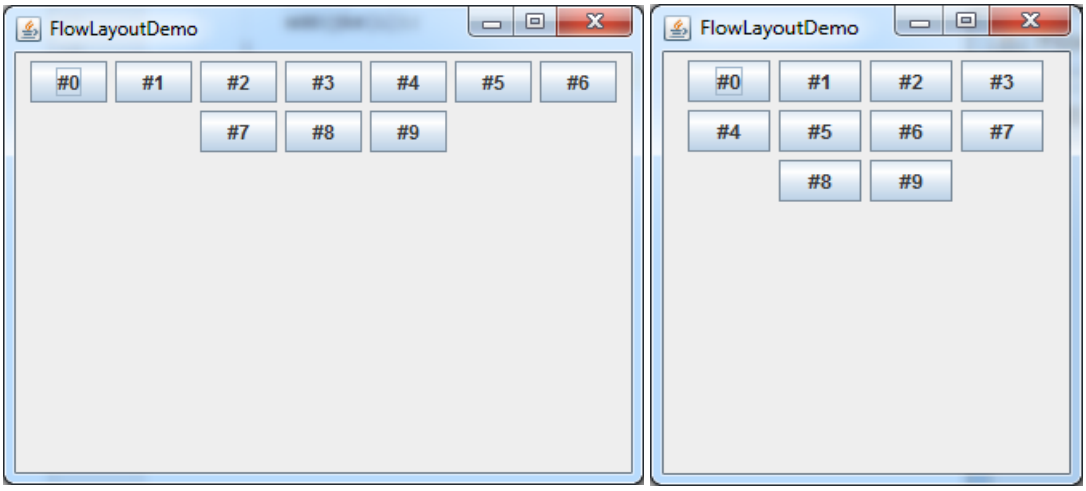

Рисунок 9. FlowLayout.

GridLayout

GridLayout — элементы управления выравниваются по таблице заданного размера. Все строки имеют одинаковую высоту, все столбцы — длину. Элемент управления может занимать только одну ячейку таблицы.

```
public class MyWindow extends JFrame {
    public MyWindow() {
        setBounds(500,500,400,300);
        setTitle("GridLayoutDemo");
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        JButton[] jbs = new JButton[10];
        setLayout(new GridLayout(4, 3));
        for (int i = 0; i < jbs.length; i++) {
            jbs[i] = new JButton("#" + i);
            add(jbs[i]);
        }
        setVisible(true);
    }
}
```

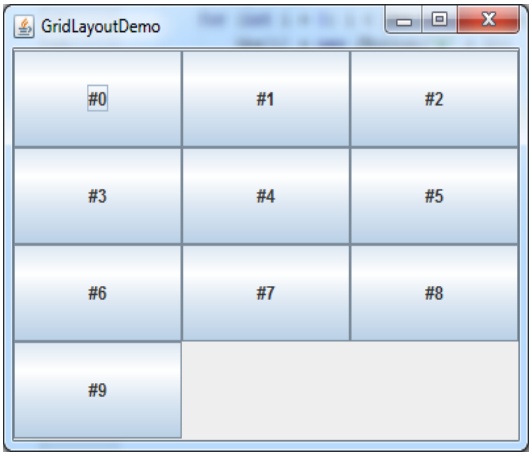

Рисунок 10. GridLayout.

Отключение компоновщика элементов — возможен вариант ручной расстановки элементов путём указания их абсолютных координат и размеров с помощью метода setBounds(), для чего необходимо выключить компоновщик элементов через setLayout(null).

Пример использования базовых элементов интерфейса:

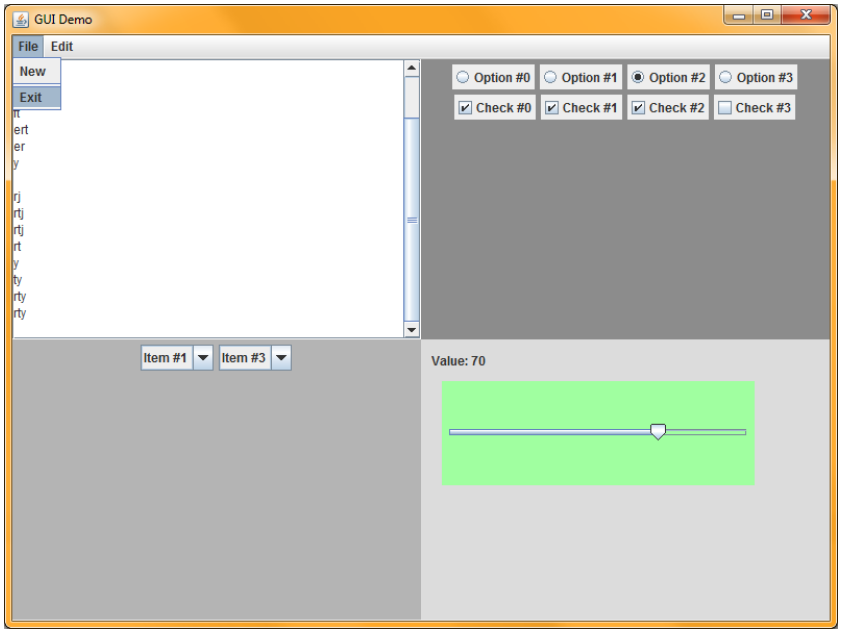

Рисунок 11. Использование различных элементов интерфейса

В первом блоке кода настраиваем параметры окна и создаём четыре панели для размещения элементов, при этом задаём им отличный друг от друга цвет. Располагаем эти панели на форме в виде таблицы (2, 2), с помощью GridLayout.

```
setBounds(500, 200, 800, 600);
setTitle("GUI Demo");
setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
setLayout(new GridLayout(2, 2));
JPanel[] jp = new JPanel[4];
for (int i = 0; i < 4; i++) {
    jp[i] = new JPanel();
    add(jp[i]);
    jp[i].setBackground(new Color(100 + i * 40, 100 + i * 40, 100 + i * 40));
}
```

На первой панели расположено многострочное текстовое поле, которое находится внутри элемента JScrollPane, что позволяет пролистывать контент этого поля.

```
jp[0].setLayout(new BorderLayout());
JTextArea jta = new JTextArea();
JScrollPane jsp = new JScrollPane(jta);
jp[0].add(jsp);
```

Во второй панели содержится 2 типа элементов: JCheckBox и JRadioButton. JCheckBox предназначен для вкл/выкл каких-либо опций (флажков), при этом одновременно может быть включено несколько JCheckBox. JRadioButton предоставляет выбор только одного пункта из набора, т.е. может быть выбрана только одна опция. Для корректной работы связанных RadioButton их необходимо заносить в ButtonGroup.

```
jp[1].setLayout(new FlowLayout());
JRadioButton[] jrb = new JRadioButton[4];
ButtonGroup bgr = new ButtonGroup();
for (int i = 0; i < jrb.length; i++) {
    jrb[i] = new JRadioButton("Option #" + i);
    bgr.add(jrb[i]);
    jp[1].add(jrb[i]);
}
JCheckBox[] jcb = new JCheckBox[4];
for (int i = 0; i < jcb.length; i++) {
    jcb[i] = new JCheckBox("Check #" + i);
    jp[1].add(jcb[i]);
}
```

На третьей панели расположена пара элементов типа JComboBox, которые представляют собой выпадающие списки. ActionListener для JComboBox проверяет событие выбора пользователем одного из пунктов.

```
jp[2].setLayout(new FlowLayout());
String[] comboStr = {"Item #1", "Item #2", "Item #3", "Item #4"};
JComboBox<String> jcombo1 = new JComboBox<String>(comboStr);
JComboBox<String> jcombo2 = new JComboBox<String>(comboStr);
jp[2].add(jcombo1);
jp[2].add(jcombo2);
jcombo1.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println(jcombo1.getSelectedItem().toString());
    }
});
```

Четвёртая панель представляет собой пример расстановки элементов с использованием абсолютных координат. На ней расположено обычное нередактируемое текстовое поле, которое показывает значение, выбранное на JSlider.

```
jp[3].setLayout(null);
JSlider js = new JSlider();
JLabel jlab = new JLabel("Value: 50");
js.setMaximum(100);
js.setMinimum(0);
js.setValue(50);
jp[3].add(jlab);
jp[3].add(js);
js.addChangeListener(new ChangeListener() {
    @Override
    public void stateChanged(ChangeEvent e) {
        jlab.setText("Value: " + js.getValue());
    }
});
jlab.setBounds(10,10,100,20);
js.setBounds(20, 40, 300, 100);
js.setBackground(new Color(160,255,160));
```

Последним пунктом идёт создание верхнего меню окна. Для этого создаются элементы JMenuBar -> JMenu -> JMenuItem. Как видно из кода ниже, для обработки нажатия на один из подпунктов меню достаточно «повесить» на него ActionListener.

```
JMenuBar mainMenu = new JMenuBar();
JMenu mFile = new JMenu("File");
JMenu mEdit = new JMenu("Edit");
JMenuItem miFileNew = new JMenuItem("New");
JMenuItem miFileExit = new JMenuItem("Exit");
setJMenuBar(mainMenu);
mainMenu.add(mFile);
mainMenu.add(mEdit);
mFile.add(miFileNew);
mFile.addSeparator(); // разделительная линия в меню
mFile.add(miFileExit);

miFileExit.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
});
addWindowListener(new WindowAdapter() {
    @Override
    public void windowClosing(WindowEvent e) {
        System.out.println("BYE");
    }
});
```

Обработка событий

Обработка событий является неотъемлемой частью разработки прикладных программ с графическим пользовательским интерфейсом (ГПИ). Любая прикладная программа с ГПИ выполняется под управлением событий, большинство которых направлено на взаимодействие с пользователем. Существует несколько типов событий, включая генерируемые мышью, клавиатурой, различными элементами интерфейса. Рассмотрим некоторые варианты обработки событий.

Обработка кликов по кнопке

```
JButton button = new JButton("Button");
add(button);
button.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button pressed...");
    }
});
```

Для отслеживания кликов по кнопке необходимо добавить ActionListener, как показано выше. Как только произойдет событие нажатия этой кнопки, выполнится метод actionPerformed().

Обработка нажатия кнопки Enter в текстовом поле

```
JTextField field = new JTextField();
add(field);
field.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("Your message: " + field.getText());
    }
});
```

При работе с текстовым полем (TextField) ActionListener отлавливает нажатие кнопки Enter, конечно только в случае, если поле находится в фокусе. Поэтому нет необходимости отслеживать именно нажатие кнопки Enter, например, через addKeyListener(…) с указанием кода этой клавиши.

Отслеживание кликов мыши

```
JPanel panel = new JPanel();
add(panel);
panel.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseReleased(MouseEvent e) {
        System.out.println("MousePos: " + e.getX() + " " + e.getY());
    }
});
```

В приведённом примере отслеживание отжатия кнопки мыши над объектом типа JPanel. В объекте типа MouseEvent попадает вся информацию о произошедшем событии, в том числе, координатах курсора в системе координат панели, кнопке, которая была нажата (левая или правая), количестве кликов (одинарный, двойной) и т.д.

Отслеживание момента закрытия окна

```
addWindowListener(new WindowAdapter() {
    @Override
    public void windowClosing(WindowEvent e) {
        System.out.println("Bye");
    }
});
```

Чтобы выполнить какое-либо действие при закрытии окна, необходимо к объекту типа JFrame добавить WindowListener. В примере выше ссылка на объект отсутствует, так как метод прописан в конструкторе класса, наследуемого от JFrame.

Список литературы (глава 3).

– Кей С. Хорстманн, Гари Корнелл. Java. Библиотека профессионала. Том 1. Основы // Пер. с англ. — М.: Вильямс, 2014. — 864 с.

– Брюс Эккель. Философия Java // 4-е изд.: Пер. с англ. — СПб.: Питер, 2016. — 1 168 с.

– Г. Шилдт. Java 8. Полное руководство // 9-е изд.: Пер. с англ. — М.: Вильямс, 2015. — 1 376 с.

Листинг 1.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

/**
 * Simple Java program to connect to MySQL database running on localhost and
 * running SELECT and INSERT query to retrieve and add data.
 * @author Javin Paul
 */
public class JavaToMySQL {

    // JDBC URL, username and password of MySQL server
    private static final String url = "jdbc:mysql://localhost:3306/test";
    private static final String user = "root";
    private static final String password = "root";

    // JDBC variables for opening and managing connection
    private static Connection con;
    private static Statement stmt;
    private static ResultSet rs;

    public static void main(String args[]) {
        String query = "select count(*) from books";

        try {
            // opening database connection to MySQL server
            con = DriverManager.getConnection(url, user, password);

            // getting Statement object to execute query
            stmt = con.createStatement();

            // executing SELECT query
            rs = stmt.executeQuery(query);

            while (rs.next()) {
                int count = rs.getInt(1);
                System.out.println("Total number of books in the table : " + count);
            }

        } catch (SQLException sqlEx) {
            sqlEx.printStackTrace();
        } finally {
            //close connection ,stmt and resultset here
            try { con.close(); } catch(SQLException se) { /*can't do anything */ }
            try { stmt.close(); } catch(SQLException se) { /*can't do anything */ }
            try { rs.close(); } catch(SQLException se) { /*can't do anything */ }
        }
    }

}
```