**Лабораторная работа №1  
Технология Java RMI**

*Целью* данной лабораторной работы является: освоить технологию RMI и выполнить удалённый вызов методов.

*Remote method Invocation* — механизм, который позволяет вызывать метод удалённого объекта. Согласно ему, все операции по подготовке и передаче данных инкапсулируются в вызываемом методе клиентского объекта-заглушки (stub). Сам же вызов метода ничем не отличается от вызова метода обычного локального объекта, за небольшим исключением:

* локальные объекты передаются по значению (копии)
* при передаче удалённого (Remote) объекта, если он экспортирован, передаётся stub этого объекта
* передаваемые объекты должны быть Serializable
* кроме всех прочих исключительных ситуаций, при вызове удалённого метода может возбуждаться исключение RemoteException (ошибки маршализации/демаршализации, передачи данных и другие возможные ошибки протокола)

Так же нужно отметить, что при вызове метода мы работаем с удалённым интерфейсом, а не с удалённым классом.

Задача RMI это организовать клиент-серверное приложение, при этом основную часть работу перенеся на сервер, разгрузив клиента. Данная технология удобна тем, что не надо беспокоится о передаче и предварительной обработке данных.

*Задача* Клиент удаленно вызывает метод сервера, который может быть запущен на удаленном компьютере. На клиенте выбирается страна, а затем город, в который хочет поехать клиент. Сервер возвращает информацию о городе, выводя ее на экран.

*Скриншоты*

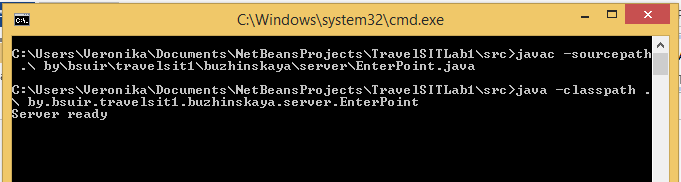


Рисунок 1 – Работа сервера

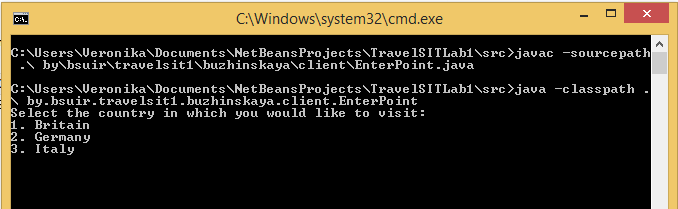


Рисунок 2 – Работа клиента

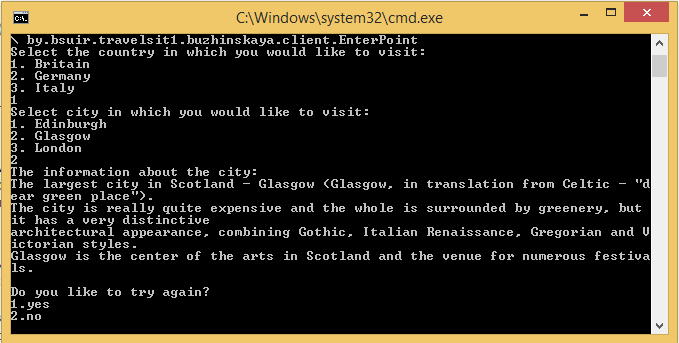


Рисунок 3 – Работа клиента

P.S. файлы (startClient.bat и startServer.bat) для запуска программы находятся в папочке build->classes

**Лабораторная работа №2  
Запуск Apache Tomcat**

*Целью* данной лабораторной работы является: запуск Apache Tomcat

*Tomcat* – [контейнер сервлетов](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%82%D0%B5%D0%B9%D0%BD%D0%B5%D1%80_%D1%81%D0%B5%D1%80%D0%B2%D0%BB%D0%B5%D1%82%D0%BE%D0%B2) с открытым исходным кодом. Реализует спецификацию [сервлетов](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%BB%D0%B5%D1%82" \o "Сервлет) и спецификацию [JavaServer Pages](https://ru.wikipedia.org/wiki/JavaServer_Pages" \o "JavaServer Pages) ([JSP](https://ru.wikipedia.org/wiki/JSP)) и [JavaServer Faces](https://ru.wikipedia.org/wiki/JavaServer_Faces" \o "JavaServer Faces) ([JSF](https://ru.wikipedia.org/wiki/JSF)). Написан на языке [Java](https://ru.wikipedia.org/wiki/Java" \o "Java).

Tomcat являясь для веб-приложений контейнером, позволяет их запускать.

Работает он следующим образом: на сервер поступает запрос от клиента. Этот запрос обрабатывает Tomcat, определяя какому приложению он назначался, и обработав его, отправляет далее по назначению. Веб-приложение, обработав запрос, формирует ответ, и ответ также проходит через Tomcat, который, проделав некоторые манипуляции, отправляет результат клиенту.

В приложение сделанному по примеру Блинова содержится две JSP странички (должно быть конечно три). Одна страничка просит ввести логин и пароль и, нажав на кнопку «login», при верном логине и пароле переправляет клиента в систему, а если нет, то перенаправляет на страничку, где оповещается об ошибке. (в данном примере я сразу переношу на страничку с оповещение об ошибке). В приложение о библиотека всё реализовано по-человечески. На первой странице создаются бины типов «Login» и «Password» по нажатию кнопки передается запрос на сервер вместе с введёнными логином и паролем , потом всё это отправляется на сервлет, в методе doPost() из объекта request достаются параметры, содержащий данные логин и пароль, обрабатываются, в базе данных ищется пользователь с данными логином и паролем и если такой есть, то он запихивается в сессию и request перенаправляется на страницу, которая показывает главную страничку данного пользователя, а если же данного пользователя нет, то request перенаправляется на страницу, где оповещается об ошибке.

*Скриншоты*

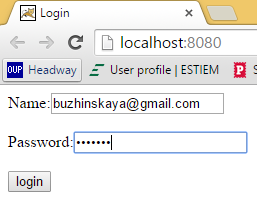


Рисунок 1 – Набор логина и пароля на первой страничке примера

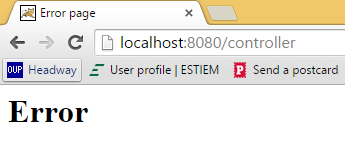


Рисунок 2 – Сообщение об ошибке

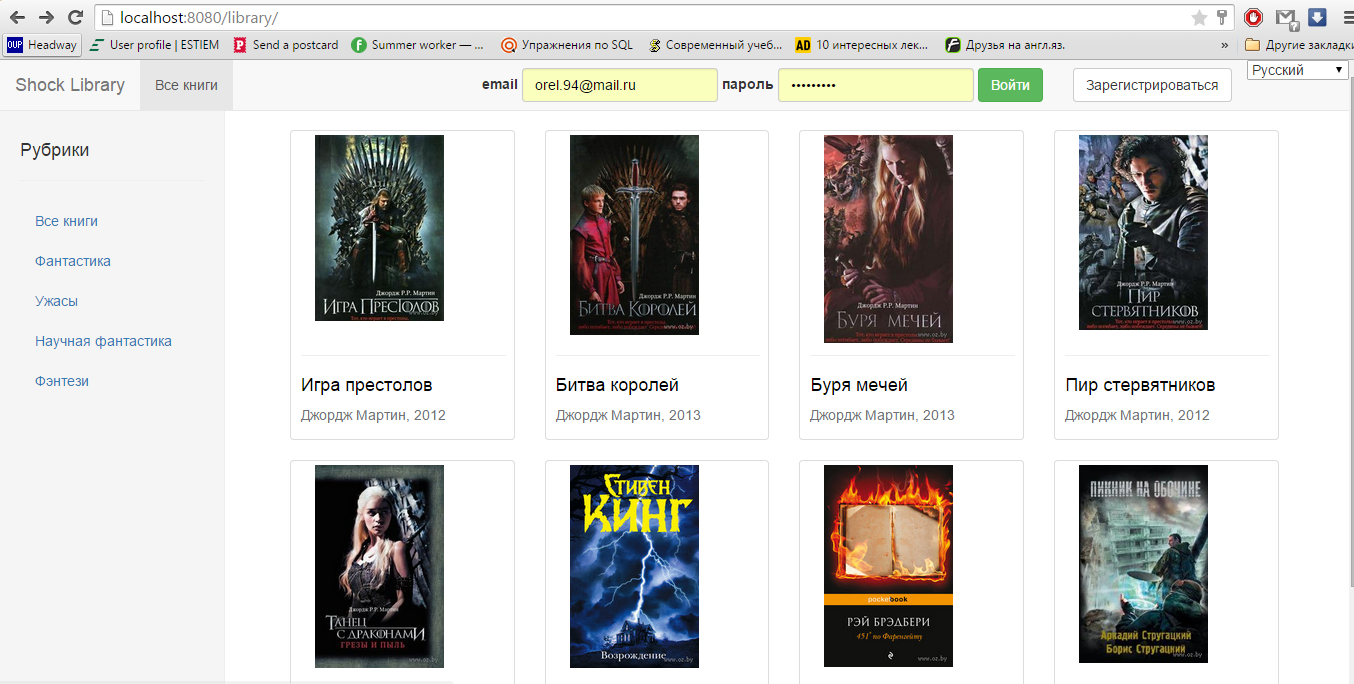


Рисунок 3 – Вход в программу

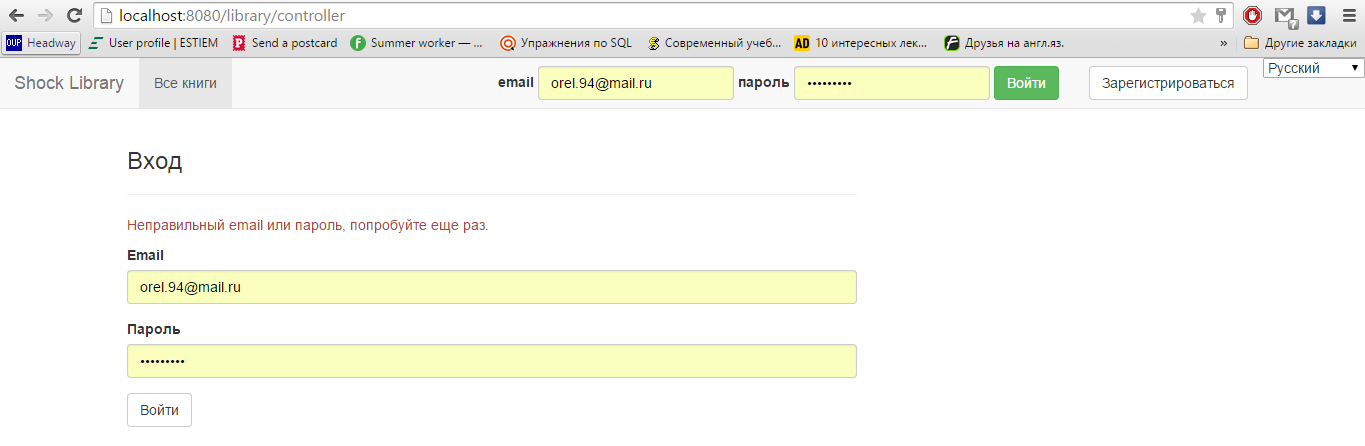


Рисунок 4 – Ошибка. Данные введены некорректно

**Лабораторная работа №3  
JPA, паттерн «Command»**

*Целью* данной лабораторной работы является: изучить и использовать JPA и паттерн «Command»

Java Persistence AP предоставляет стандартный механизм для ORM, EntityManager API для создания, обновления и удаления объектов и язык запросов JPA-QL или JPQL для извлечения сущностей, запросы которого могут быть проверены на корректность на этапе компиляции и могут быть динамически сформированы на этапе выполнения приложения [1].

Данная технология помогает соотносить классы сущностей с соответствующими таблицами в базе данных, а также предоставляетAPI для манипулирования сущностями. API позволяет сохранять, восстанавливать, обновлять или удалять объекты, освобождая программиста от написания JDBC и SQL кода. Такжепредоставляетязык запросов для восстановления объектов из базы данных. Это один из наиболее важных элементов, так как некорректные SQL утверждения могут замедлить базу данных. Этот подход также освобождает приложение от внутренних SQL, разбросанных по всему приложению.

Данная технология очень удобна, так как при минимуме усилий создает управление базой данных с любой структурой связей между сущностями.

Что касается паттерна «Command», то он преобразовывает запрос на выполнение действия в отдельный объект-команду. Такая инкапсуляция позволяет передавать эти действия другим функциям и объектам в качестве параметра, приказывая им выполнить запрошенную операцию. Команда – это объект, поэтому над ней допустимы любые операции, что и над объектом.

Смысл использования паттерна «Command» в данной лабораторной работе заключается в разделении выполнении определенной последовательности команд и непосредственного содержания данных команд, кроме этого паттерн помогает избежать использования команды типа «switсh-case» при выборе команды, упрощая тем самым код программы. Также программа становится легко расширяемой, ибо для добавления новой команды не требуется переписывать и/или редактировать старый код, а просто создать новый класс, реализующий интерфейс, и зарегистрировать этот класс каким-то образом.

В данной лабораторной работе приведёт пример реализации использования паттерна «Command». От клиента посылаются определенные команды, которые распознаются на сервере. Далее создаются соответствующие экземпляры класса команды и выполняются его методы.

Далее ответим на ряд вопрос которые были заданы при выдачи данной лабораторной работы:

1. Что такое CRUD?

CRUD – это сокращённое именование четырёх базовых функций, используемых при работе с базами данных, а именно: create (создание), read (чтение), update (редактирование/обновление), delete (удаление).

1. Что такое ORM?

ORM (Object-relational mapping) – это прослойка между базой данных и кодом, которая позволяет созданные в программе объекты складывать/получать в/из базы данных.

1. Что такое method redirect?

Method redirect – это метод, который перенаправляет клиента на новый адрес URL ( задаёт новый адрес и условия прекращения выполнения текущей страницы.

1. Что такое persistence.xml?

Файл persistence.xml описывает блоки хранения. Он представляет собой файл описания хранения с использованием Java Persistence API (JPA). Он применяется для объявления следующих объектов:

* Управляемые классы хранения.
* Связь между объектом и отношением.  
  JPA предлагает несколько способов связывания классов java с таблицами в реляционной базе данных.
* Информация о конфигурации администраторов объектов и классов фабрики администраторов объектов.

1. Что такое Connectionpool?

Connectionpool – это хранилище конекшенов. Он подразумевает, что в нашем распоряжении имеется некоторый пул-соединений к базе данных. Когда новый пользователь запрашивает доступ к БД, ему выдаётся уже открытое соединение из этого пула, если нет свободного соединения, то пользователь ждёт его.

**Лабораторная работа №4  
Создание приложения J2EE с помощью EJB 3.1**

*Целью* данной лабораторной работы является: изучить принцип асинхронной обработки запроса

Начиная с версии 7.0 в Tomcat в полной мере реализована поддержка Servlet 3.0, а вместе с ней поддержка асинхронных сервлетов.

Суть принципа асинхронной обработки запроса заключается в том, что клиент открывает долгоживущее HTTP-соединение, которое хранится на сервере до того момента, пока сервер не будет готов отослать ответ обратно. Посылка ответа инициируется сервером. Такой подход позволяет избавиться от многократного опроса сервера множеством клиентов с целью получить как можно более свежую информацию. При использовании асинхронных сервлетов соединение открывается один раз, и потом сервер сам отошлет ответ, когда будет что отсылать. Естественно, когда с сервера придет ответ, нужно установить соединение заново.

В данной лабораторной работе при добавлении сообщения в базу данных используется JMS. Так как он работает асинхронно, то порой возникает ситуация, что при добавлении сообщения и переходу к списку всех сообщений, новое в списке не появляется.

*Скиншот*



Рисунок 1 – Работа приложения

**Лабораторная работа №5-6**

**EJB + JSF**

Ответим на ряд вопрос на которые необходимо было ответить после выполнения примера:

1. Как сконфигурирован JPA?

Основной конфигурационной единицей JPA является Persistence Unit – в нем описывается тип провайдера, который предоставляет “услуги” JPA, а также список классов, за которые он “отвечает” – JPA-сущностей. В блоке <properties> описываются параметры доступа к базе данных, её тип, настройки логирования и тд.

После выхода спецификации WebServlet 3.0 часть описания может быть расположена непосредственно в классах в виде атрибутов методов, классов, полей.

1. Чем отличается JSF от JSP?

JavaServer Faces(JSF)— это фреймворк для веб-приложений на Java. Он служит для облегчения разработки пользовательских интерфейсов для Java EE приложений. Подход JSF основывается на использовании компонентов. Состояние компонентов пользовательского интерфейса сохраняется, когда пользователь запрашивает новую страницу и затем восстанавливается, если запрос повторяется. MVC: Model-entity или др классы; View – jsf страница в xhtml формате; Сontroller–managed bean, который интерпретирует данные, введённые пользователем, и информирует модель о необходимости соответствующей реакции.

Технология Java Server Pages (JSP) создана, чтобы облегчить создание

страниц с динамическим содержанием. В то время как сервлеты наилучшим образом подходят для выполнения контролирующей функции приложения в виде обработки запросов и определения содержания и вида ответа, страницы JSP выполняют функцию отображения

результатов работы приложения в виде текстовых документов типа HTML, XML, WML и некоторых других. JSP поддерживают как JavaScript, так и HTML-теги. JavaScript обычно используется, чтобы добавить функциональные возможности на уровне HTML-страницы.

1. Что такое dependency injection и inversion of control?

Inversion of Control (инверсия управления) — это некий абстрактный принцип, набор рекомендаций для написания слабо связанного кода. Суть которого в том, что каждый компонент системы должен быть как можно более изолированным от других, не полагаясь в своей работе на детали конкретной реализации других компонентов.

Dependency Injection (внедрение зависимостей) — это одна из реализаций этого принципа (помимо этого есть еще Factory Method, Service Locator).

1. В чем отличие stateless от statfull beans?

Stateful может хранить какое-либо состояние между запросами, соответственно контейнер имеет пул таких бинов и для каждой сессии выделяет отдельный экземпляр. Stateless никакого состояния не хранит, а, следовательно, может использоваться один и тот же экземпляр

P.S. Лабораторные работы 2-3 выполнены в IDEA, потому что мне так привычнее, т.к. идея много чего выполняет автоматически и там понятно где, что происходит.