МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Учреждение образования
«Витебский государственный технологический университет»

КОМПОНЕНТНОЕ ПРОГРАММИРОВАНИЕ

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

для слушателей специальности 1-40 01 73 «Программное обеспечение информационных систем» вечерней и заочной форм обучения

Витебск
2021

УДК 681.5 (075.8)

Составители:

В. Е. Казаков, П. Г. Деркаченко

Рекомендовано к изданию редакционно-издательским советом
УО «ВГТУ» протокол № 9 от 28.05.2021.

Компонентное программирование : лабораторный практикум / сост. В. Е. Казаков, П. Г. Деркаченко. – Витебск : УО «ВГТУ», 2021. – 34 с.

Издание является руководством для выполнения лабораторных работ по курсу «Технология компонентного программирования», позволяет получить практические навыки в области компонентного программирования, закрепить знания, полученные на теоретической части курса, а также ознакомиться с возможностями фреймворка Spring.

УДК 681.5 (075.8)

СОДЕРЖАНИЕ

Лабораторная работа 1
Установка и настройка среды разработки Eclipse

Установка Eclipse IDE

1. Скопировать Eclipse IDE из папки «atpp\It-задания\ВСРПП\2 семестр\development» в папку «C:\www»

2. После запуска Eclipse установить в качестве рабочей папки (eclipse workspace) папку H:\eclipse, предварительно её создав.

3. Задать расположение jdk, использовав команды верхнего меню IDE: «Window –> Preferences –> Java –> Installed JREs –> Add» –> «Standard VM –> Next». В появившемся окне указать JRE home – путь к установленной JDK-8.

4. Нажать «OK», затем – «Finish»

5. Установить выбранную jdk по умолчанию (рис. 1.1).

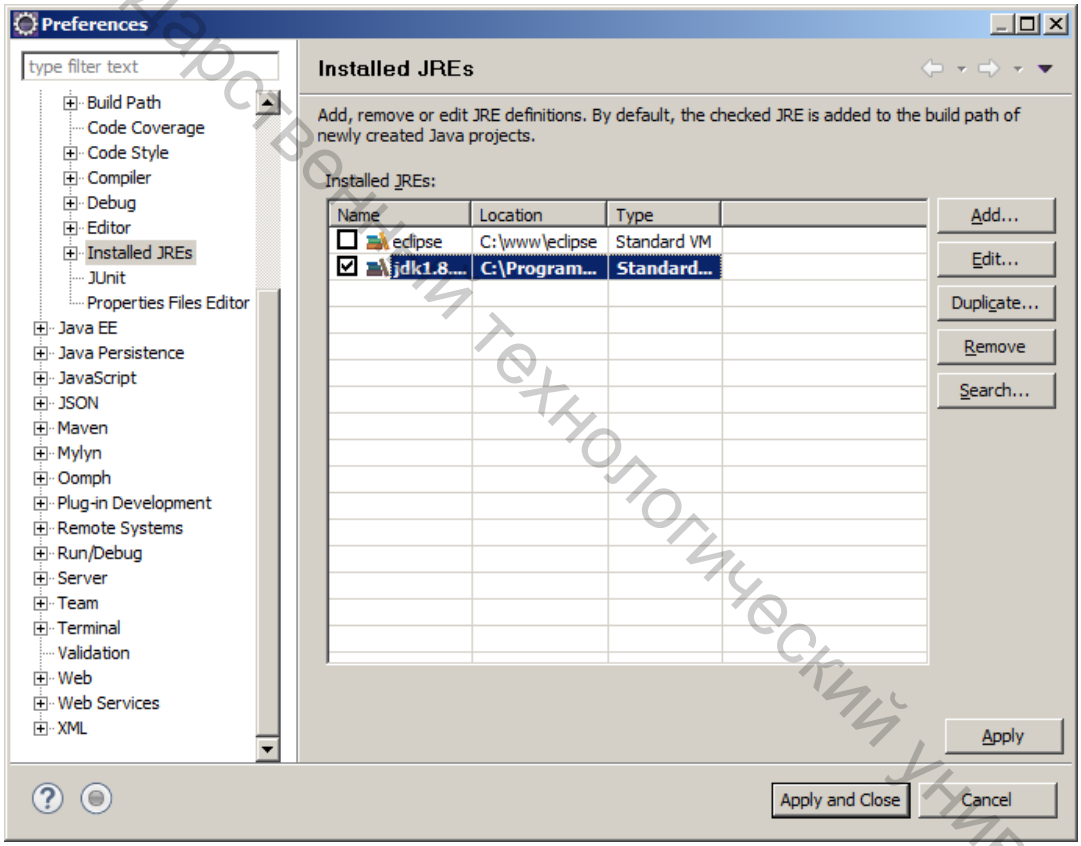

Рисунок 1.1 – Выбор JDK по умолчанию

6. Нажать «Apply and Close».

Настройка прокси-сервера в Eclipse IDE

1. В верхнем меню IDE Eclipse выбрать команды Window –> Preferences –> General –> Network Connections.

2. В раскрывающемся списке «Active Provider» выбрать «Manual» (рис. 1.2).

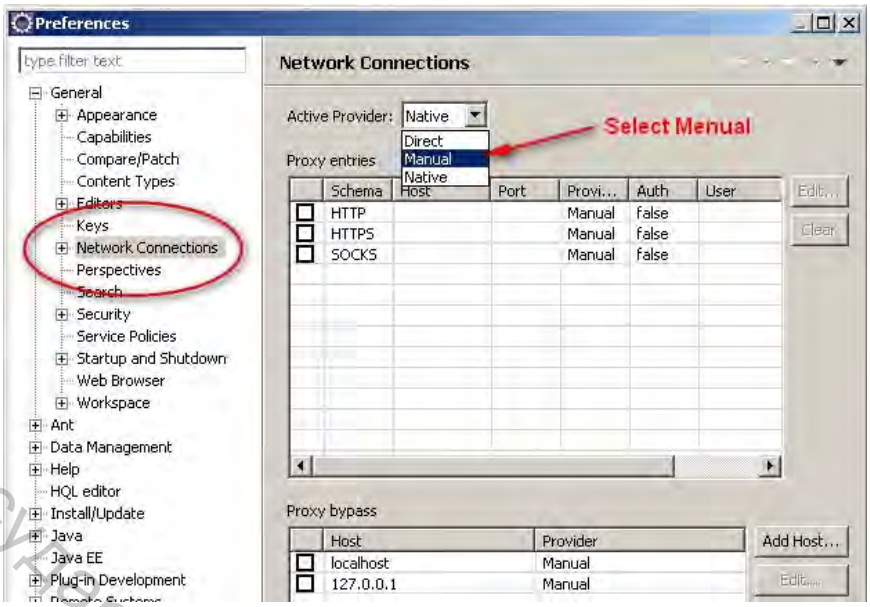

Рисунок 1.2 – Настройка интернет-соединения в IDE Eclipse

3. Выбрать Http в списке и нажать кнопку «Edit».

4. Ввести хост прокси-сервера и номер порта, а также пользовательские сетевые имя пользователя и пароль (рис. 1.3).

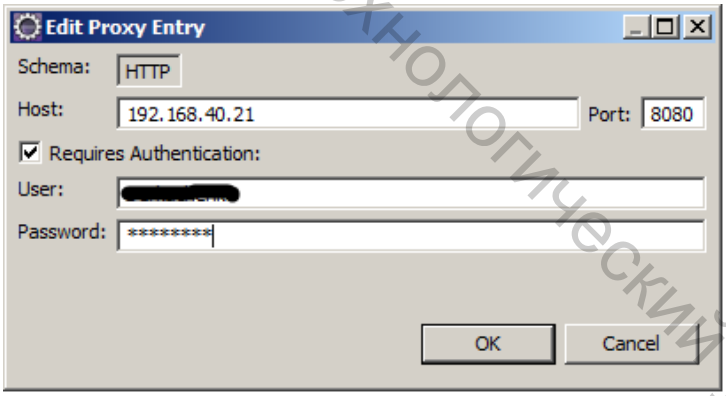

Рисунок 1.3 – Окно настройки прокси-сервера в IDE Eclipse

5. Нажать «ОК».

Имя хоста и порта может быть иным. Они должны совпадать с соответствующими настройками интернет-соединения.

6. Повторить то же для Https.

7. Выбрать в том же окне (рис. 1.2) команды Security –> Security storage.

8. Выбрать пункт «Windows Integration» и нажать кнопку «Change Password».

9. Нажать кнопку «**No**», затем – «ОК».

10. Нажать кнопку «Apply and Close».

Настройка Maven в Eclipse IDE

1. В верхнем меню IDE Eclipse выбрать команды Help –> Install New Software» ...

2. Нажать кнопку «Add» в правом верхнем углу.

3. Во всплывающем окне ввести имя – M2Eclipse и расположение – http://download.eclipse.org/technology/m2e/releases (рис. 1.4).

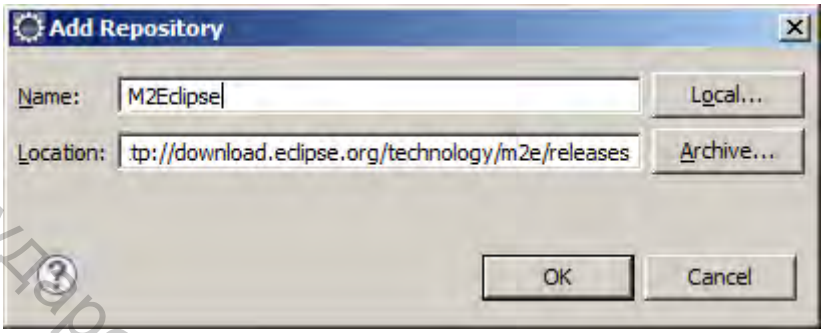

Рисунок 1.4 – Сетевой ввод адреса плагина Maven

4. Следовать инструкциям установщика.

5. Перезапустить Eclipse.

Настройка прокси-сервера maven

1. Скопировать папку .m2 из atpp\It-задания\ВСРПП\2 семестр\ProxyEclipse\ в свою область на диске С (например, C:\Users\12345).

2. Указать в разделе «proxies» файла settings.xml (находится в папке .m2) свои логин и пароль, а также хост и порт прокси-сервера, которые были введены ранее при настройке интернет-соединения IDE Eclipse.

3. Открыть Eclipse и перейти в настройки Window –> Preferences–> Maven–> User Settings.

4. Нажать кнопку «Browse» в разделе «User Settings» и выбрать путь к файлу settings.xml.

5. Далее – нажать последовательно кнопки «Update settings» и «Apply and Close».

Работа с демонстрационным приложением

1. Скачать демонстрационное приложение spring-boot-web-jsp из папки atpp\It-задания\ВСРПП\2 семестр\ProxyEclipse и разместить его в рабочей области Eclipse.

2. В Eclipse выбрать команды File–> Import –> Existing maven project.

3. Нажать кнопку Next.

4. В появившемся окне нажать Browse и выбрать приложение spring-boot-web-jsp из рабочей области Eclipse. Нажать «ОК», затем – «Finish»

5. Собрать проект:

a. После того, как проект появится в менеджере проектов, кликнуть по нему ПКМ и выбрать команды Maven –> Update Project.

b. В появившемся окне поставить галочку напротив пункта Force Update.

c. Нажать «ОК».

6. Запустить проект:

a. Кликнуть по проекту ПКМ, выбрать команды Run As –> Java Application

b. В появившемся списке классов выбрать класс SpringBootWebApplication – com.mkyong.

c. Нажать «ОК».

После запуска приложение будет доступно по адресу http://localhost:8088.

Задание к лабораторной работе 1

Настроить на своей рабочей станции IDE Eclipse для разработки компонент веб-приложения. Убедиться в корректности настроек, запустив контрольный пример.

Лабораторная работа 2
Разработка компонентного приложения с простой архитектурой

Архитектура ПО – набор ключевых правил, определяющих организацию системы:

– совокупность структурных элементов системы и связей между ними;

– поведение элементов системы в процессе их взаимодействия;

– иерархию подсистем, объединяющих структурные элементы.

Сложная программная система должна быть разделена на небольшие подсистемы, каждую из которых можно разрабатывать независимо (в какой-то степени) от других.

Структура ПО должна удовлетворять следующим требованиям:

– каждая подсистема должна **инкапсулировать** свое содержимое (скрывать его от других подсистем);

– каждая подсистема должна иметь четко определенный **интерфейс** с другими подсистемами.

Декомпозиция является главным способом преодоления сложности разработки ПО. Принципы декомпозиции:

– слабая связанность – low coupling (количество связей между отдельными подсистемами должно быть минимальным);

– сильное сцепление – high cohesion (связность отдельных частей внутри каждой подсистемы должна быть максимальной).

Подходы к декомпозиции:

– функционально-модульный – описывается в терминах иерархии ее функций и иерархии структур данных;

– объектно-ориентированный – описывается в терминах объектов и связей между ними, а поведение описывается в терминах обмена сообщениями между объектами.

Основные принципы проектирования:

– разделение функций – разделение приложение на отдельные компоненты с, по возможности, минимальным перекрытием функциональности. Важным фактором является предельное уменьшение количества точек соприкосновения, что обеспечит high cohesion и low coupling;

– принцип единственности ответственности – каждый компонент или модуль должен отвечать только за одно свойство/функцию или совокупность связанных функций;

– принцип минимального знания (также известный как Закон Деметера (Law of Demeter, LoD)). Компоненту или объекту не должны быть известны внутренние детали других компонентов;

– избегание повторов (Don't repeat yourself, DRY) – определенная функциональность должна быть реализована только в одном компоненте и не должна дублироваться ни в одном другом компоненте;

– минимизация проектирования наперед (big design upfront, BDUF) – проектируйте только то, что необходимо. Этот принцип называют YAGNI («You ain't gonna need it»).

– принцип открытия/закрытия (OCP: Open/Closed Principle) – объекты проектирования (классы, функции, модули и т.д.) должны быть открыты для расширения, но закрыты для модификации;

– принцип замещения Лисков (LSP: Liskov Substitution Principle) – функции, которые используют ссылки на базовые классы, должны иметь возможность использовать объекты производных классов, не зная об этом;

– принцип изоляции интерфейса (ISP: Interface Segregation Principle) – клиент не должен вынужденно зависеть от элементов интерфейса, которые он не использует;

– принцип обращения зависимости (DIP: Dependency Inversion Principle) – модули верхних уровней не должны зависеть от модулей нижних уровней. Оба типа модулей должны зависеть от абстракций. Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций.

Методика выполнения задания

1. Запустить IDE Eclipse.

2. В верхнем меню выбрать команду File –> New –> Maven Project, в появившемся окне выбрать следующие пункты (рис. 2.1).

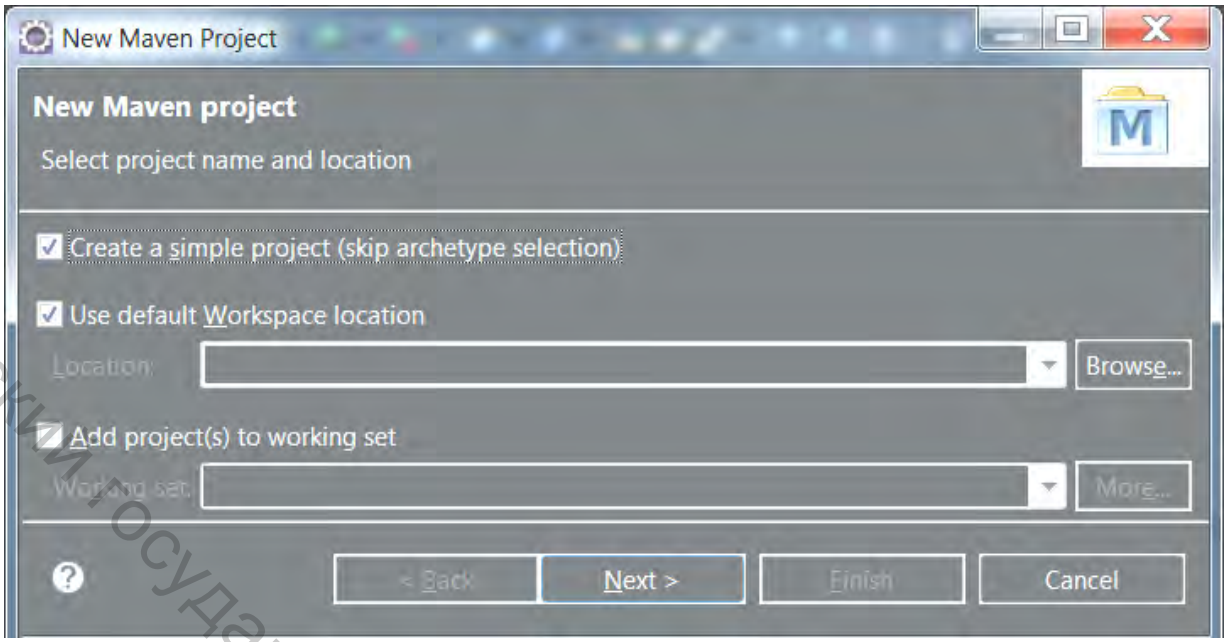

Рисунок 2.1 – Окно создания проекта

3. Нажать Next.

4. Заполнить появившееся окно (рис. 2.2).

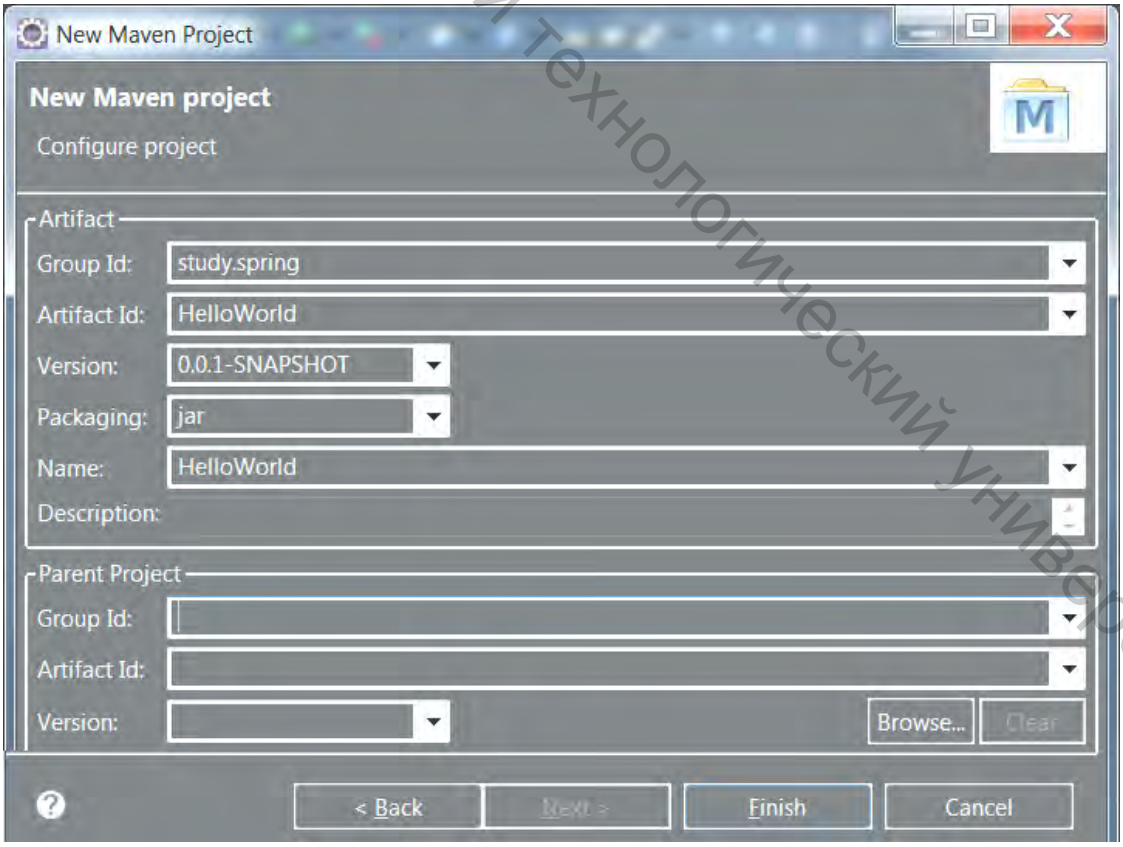

Рисунок 2.2 – Окно настройки проекта

5. Нажать Next.

6. В окне Project explorer'а (левая панель IDE) кликнуть ПКМ по папке src/main/java проекта и выбрать команды New –> Package. В появившемся окне в графу Name ввести study.spring и нажать Finish.

7. В Project explorer'е найти файл pom.xml и преобразовать его к следующему виду:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns: xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>study.spring</groupId>
  <artifactId>HelloWorld</artifactId>
  <version>0.0.1-SNAPSHOT</version>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.2.4.RELEASE</version>
        <relativePath/>
    </parent>
    <properties>
         <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
         <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
            <optional>true</optional>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

8. В Project explorer'е кликнуть ПКМ по названию пакета study.spring и выбрать команды New –> Class. В появившемся окне ввести данные для создания точки входа в приложение согласно рисунку 2.3.

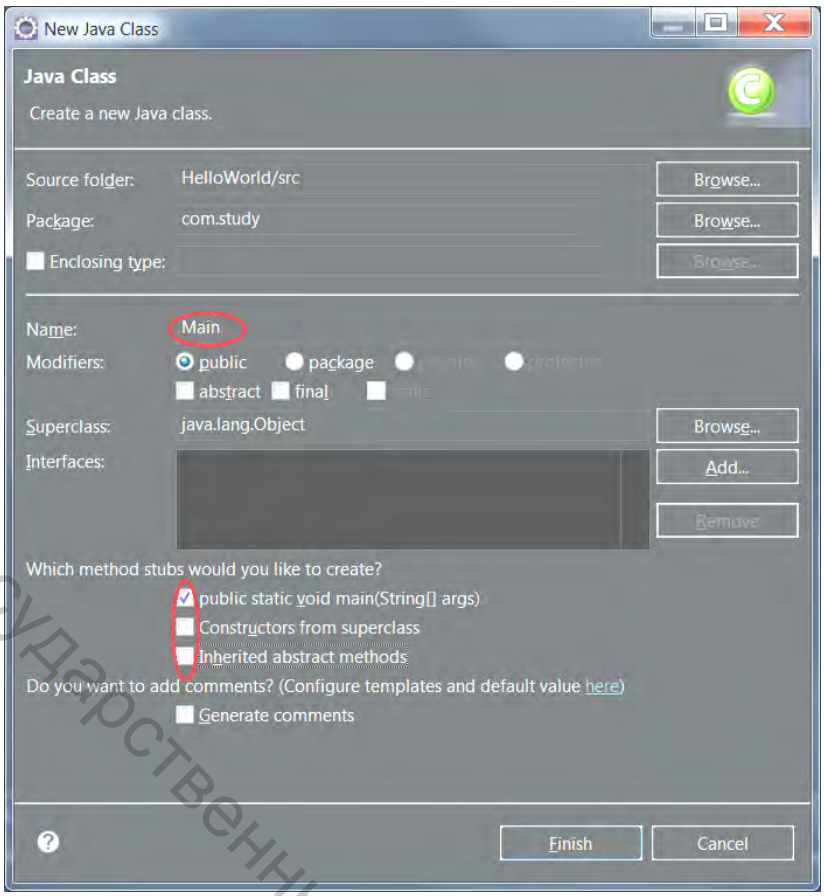

Рисунок 2.3 – Создание точки входа в приложение

9. Открыть класс Main и преобразовать его к виду:

```
package study.spring;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Main {
	public static void main(String[] args) {
		System.out.println("Hello");
		SpringApplication.run(Main.class, args);
	}
}
```

10. В окне Project explorer'а (левая панель IDE) кликнуть ПКМ по папке src/main/resources, выбрать команды New –> Other –> General –> File. Создать файл application.yml.

11. Открыть его в редакторе IDE и ввести следующий код:

```
server:
    port: 8088
```

с соблюдением всех отступов: команда server не имеет отступов от начала строки, команда port имеет отступ в 4 пробела.

12. Сохранить проект.

13. Собрать проект.

14. Запустить проект.

При успешном запуске в консоли IDE должно отобразиться 2 раза слово hello и информация о запуске веб-сервера.

15. Остановить приложение.

Добавление сервиса

1. Кликнуть ПКМ по пакету study.spring и выбрать команды New –> Package. Дать название пакету study.spring.service.

2. Нажать кнопку Finish.

3. В Project explorer'е кликнуть ПКМ по пакету study.spring.service и выбрать команды New –> Class. Ввести название класса HelloService. Открыть класс в редакторе и изменить его следующим образом:

```
@Service
public class HelloService {
	public void prn() {
		System.out.println("Hello!");
	}
}
```

4. Изменить класс Main следующим образом:

```
@SpringBootApplication
public class Main {
	public static void main(String[] args) {
		ConfigurableApplicationContext ctx = SpringApplication.run(Main.class,
args);
		HelloService s = ctx.getBean(HelloService.class);
		s.prn();
	}
}
```

5. Запустить проект.

Задание к лабораторной работе 2

Разработать сервис, запрашивающий с консоли ввод имени пользователя и затем отображающий на консоли приветствие данного пользователя.

Лабораторная работа 3
Разработка приложения с использованием IoC внедрения компонент

Контейнер Inversion of Control (IoC контейнер) основывается на шаблоны проектирования «Инверсия управления». IoC контейнер предоставляет средства конфигурирования и управления объектами Java с помощью рефлексии. Контейнер отвечает за управление жизненным циклом объекта: создание объектов, вызов методов инициализации и конфигурирование объектов путем связывания их между собой. Объекты, создаваемые контейнером, также называются управ-

ляемые объекты или beans. Конфигурирование контейнера осуществляется путем загрузки XML файлов, содержащих определение bean'ов и предоставляющих информацию, необходимую для создания bean'ов. Также оно может быть выполнено при помощи аннотаций, применяемых непосредственно в классах приложения. Объекты могут быть получены либо с помощью Поиска зависимости (Dependency Lookup, или сокращенно DL), либо Внедрения зависимости (Dependency Injection – DI). DL – шаблон проектирования, в котором вызывающий объект запрашивает у объекта-контейнера экземпляр объекта с определенным именем или определенного типа. Этот шаблон используется, когда конфигурирование контейнера производится с применением аннотаций. DI – шаблон проектирования, в котором контейнер передает экземпляры объектов по их имени другим объектам либо с помощью конструктора, либо свойства, либо фабричного метода. Данный шаблон используется, когда конфигурирование контейнера производится в XML-файлах.

На практике обычно применяют сразу оба способа конфигурирования.

Основные преимущества IoC контейнеров:

– управление зависимостями;
– упрощение повторного использования классов или компонентов;
– более «чистый» код (классы не инициализируют вспомогательные объекты).

Методика выполнения задания

Рассмотрим простое приложение по сложению двух чисел и выводу результата. Все приложение можно написать при помощи лишь одного класса Calculate. Что, если нам понадобилось изменить сложение в методе operate на умножение или выводить результат в методе showResult не на экран, а в файл? В этих случаях код программы нужно постоянно менять.

```
public class Calculate {
	public Calculate() {
	}

	public static void main(String[] args) {
		Calculate calc = new Calculate();
		calc.execute(10L, 15L);
	}

	private long operate(long op1, long op2) {
		return op1 + op2;
	}

	private String getOpsName() {
		return " plus ";
	}

	public void execute(long op1, long op2) {
		System.out.println ("The result of " + op1 + getOpsName()
+ op2 + " is " + operate(op1, op2) + "!");
	}
}
```

Необходимо разъединить логику приложения от математических операций при помощи интерфейсов. Первый интерфейс будет называться Operation. В нем мы объявим два метода operate(long op1, long op2) и getOpsName().

```
public interface Operation {
	long operate(long op1, long op2);
	String getOpsName();
}
```

После этого, если требуется сделать сложение, то можно будет просто реализовывать этот интерфейс:

```
public class OpAdd implements Operation {
	public OpAdd() {
	}

	public String getOpsName() {
		return " plus ";
	}

	public long operate(long op1, long op2) {
		return op1 + op2;
	}
}
```

Для других математических действий все делается аналогично:

```
public class OpMultiply implements Operation {
	public OpMultiply() {
	}

	public String getOpsName() {
		return " times ";
	}

	public long operate(long op1, long op2) {
		return op1 * op2;
	}
}
```

Тогда класс Calculate будет выглядеть следующим образом:

```
public class Calculate {
	private Operation ops = new OpAdd();

	public static void main(String[] args) {
		Calculate calc = new Calculate();
		calc.execute(10L, 15L);
	}

	public void execute(long op1, long op2) {
		System.out.println("The result of " + op1 + ops.getOpsName()
 + op2 + " is " + ops.operate(op1, op2) + "!");
	}
}
```

Теперь имеем OpAdd, OpMultiply и основной класс Calculate. В главном классе определяем один из Op* классов и используем его.

Если требуется изменить действия, то нужно просто поменять названия классов. Здесь есть одна небольшая проблема: код Calculate должен быть изменен или перекомпилирован, если нужно изменить математическую операцию или по-другому вывести результат. Код, который создает экземпляр OppAdd, жестко закодирован в Calculate.

Способность соединять и рассоединять повторно используемые Java бины в приложении – это важная сторона Spring framework. Суть в том, что тогда Calculate не будет работать с экземплярами Operation, а делегирует эту задачу Spring контейнеру. Spring контейнер, в свою очередь, читает конфигурационный файл и сам конфигурирует Calculate. Однако в этом случае класс Calculate должен быть аннотирован специальным образом:

```
import lab2_1.component.Operation;

@SpringBootApplication
@ComponentScan("lab2_1")
@Component
public class Calculate {
	@Autowired
	@Qualifier("add")
	private Operation ops;

	public static void main(String[] args) {
		ConfigurableApplicationContext context = SpringApplica-
tion.run(Calculate.class, args);
		Calculate calc = context.getBean(Calculate.class);
		calc.execute(10L, 15L);
		context.close();
	}

	public void execute(long op1, long op2) {
		System.out.println("The result of " + op1 + ops.getOpsName() + op2 + " is
" + ops.operate(op1, op2) + "!");
	}
}
```

Аннотация `@Component` указывает на то, что объект данного класса должен быть создан и помещен в контейнер, а `@Autowired` указывает на то, что данное поле должно быть внедрено контейнером. The `@Qualifier` annotation is used to resolve the autowiring conflict, when there are multiple beans of same type.

Компонентам, предназначенным для внедрения по одинаковому интерфейсу Operation, дается имя:

```
package lab2_1.component;

import org.springframework.stereotype.Service;

@Service("add")
public class OpAdd implements Operation {
	public OpAdd() {
	}
```

```
	public String getOpsName() {
		return " plus ";
	}

	public long operate(long op1, long op2) {
		return op1 + op2;
	}
}
```

```
package lab2_1.component;

import org.springframework.stereotype.Service;

@Service ("mult")
public class OpMultiply implements Operation {
	public OpMultiply() {
	}

	public String getOpsName() {
		return " times ";
	}

	public long operate(long op1, long op2) {
		return op1 * op2;
	}
}
```

Если для внедрения возможен только один кандидат (например, если мы захотим внедрить `Calculate`), аннотация `@Qualifier` не требуется.

Алгоритм выполнения задания

1. Создать Maven проект.

2. Указать Group ID (разработчик) – by.vstu.zamok и Artifact Id (имя проекта) – lab2.

3. Указать зависимость spring в pom.xml:

```
...
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.2.4.RELEASE</version>
        <relativePath/>
    </parent>
...

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
            <optional>true</optional>
        </dependency>
```

```
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
...
```

4. Создать интерфейс Operation, классы OpAdd и OppMultiply (пакет lab2_1.component) и класс Calculate (пакет lab2_1.runner).

5. Добавить в папку src/main/resources файл application.yml (см. лабораторную работу 2).

Структура проекта в project explorer'е должна выглядить следующим образом (рис. 3.1).

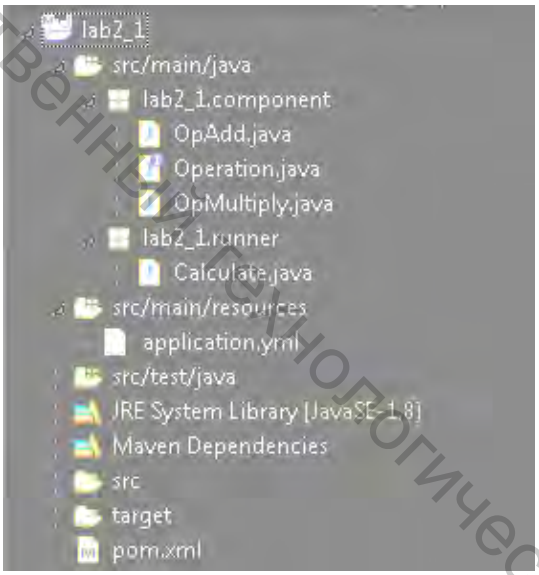

Рисунок 3.1 – Структура проекта с IoC внедрением компонент

6. Запустить приложение (рис. 3.2).

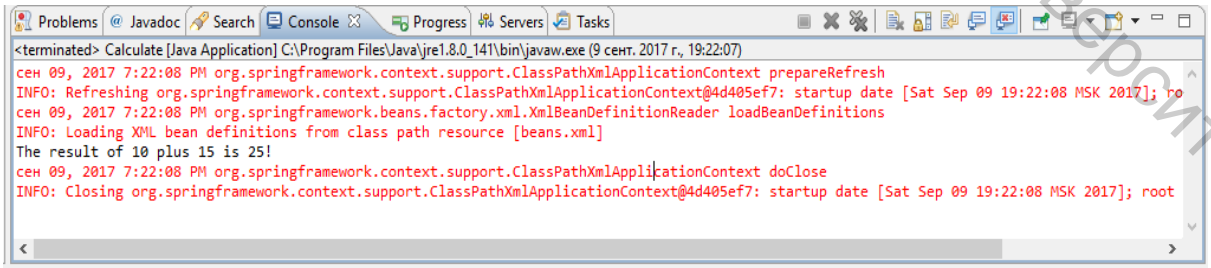

Рисунок 3.2 – Результат запуска проекта с IoC внедрением компонент

7. Изменить бин вычисления суммы на бин вычисления произведения и снова запустить приложение.

Задание к лабораторной работе 3

Добавить дополнительную математическую операцию к приложению, показанному в примере.

Вариант	Задание
1, 6, 11	Вычитание
2, 7, 12	Деление
3, 8, 13	Возведение в степень
4, 9, 14	Среднее арифметическое
5, 10, 15	Среднее геометрическое

Лабораторная работа 4
Разработка приложения с использованием фреймворка Spring boot

Spring Boot – это среда на основе Java с открытым исходным кодом, используемая для создания микросервисов. Micro Service (микросервис) – это архитектура, которая позволяет разработчикам самостоятельно разрабатывать и развертывать сервисы. Каждый работающий сервис имеет свой собственный процесс, и это обеспечивает облегченную модель для поддержки бизнес-приложений.

Микросервис предлагает своим разработчикам следующие преимущества:

1. Простое развертывание.
2. Простая масштабируемость.
3. Совместимость с контейнерами.
4. Минимальная конфигурация.
5. Меньшее время производства.

Spring Boot предлагает следующие преимущества для своих разработчиков:

1. Предоставляет гибкий способ настройки Java Beans, конфигураций XML и транзакций базы данных.
2. В Spring Boot все настраивается автоматически; ручные настройки не требуются.
3. Облегчает управление зависимостями.
4. Включает встроенный контейнер сервлетов.

Методика выполнения задания

1. Создать Maven проект.

2. Указать зависимость spring boot в pom.xml.

3. Создать пакет entity, в котором будут храниться классы-сущности.

4. Создать абстрактную сущность и её наследников – сущности, отражающие предметную область:

```
public class Group extends AbstractEntity {
	private String name;
	private List<Student> students;

	// геттеры, сеттеры
}

public class Student extends AbstractEntity {
	private String name;
	private String surname;
	private Group group;

	// геттеры, сеттеры
}
```

5. Создать пакет dao (здесь будут храниться объекты-заглушки доступа к данным). Пакет будет содержать общий интерфейс и интерфейсы для каждой из сущностей:

```
public interface Dao<T extends AbstractEntity> {
	T read(Long id);
	List<T> read();
	void save(T entity);
	void delete(Long id);

}

public interface GroupDao extends Dao<Group> {
}

public interface StudentDao extends Dao<Student> {
	List<Student> read(Group group);
}
```

6. Создать реализации DAO внутри пакета dao.

DAO должен управлять только одним видом сущностей, в сервисах же можно внедрить несколько DAO для реализации бизнес-логики, кеширования и т.д.

7. Создать пакет для сервисов.

```
public interface Service<T extends AbstractEntity> {
	T read(Long id);
	List<T> read();
	void save(T entity);
	void delete(T entity);
}

public interface GroupService extends Service<Group> {
}

public interface StudentService extends Service<Student> {
```

```
	List<Student> read(Group group);
}
```

8. Создать пакет для реализации сервисов service.impl и создать сервисы.

```
@Service
public class GroupServiceImpl implements GroupService {

	@Autowired
	private GroupDao dao;

	@Override
	public Group read(Long id) {
		return dao.read(id);
	}

	@Override
	public List<Group> read() {
		return dao.read();
	}

	@Override
	public void save(Group entity) {
		dao.save(entity);
	}

	@Override
	public void delete(Group entity) {
		dao.delete(entity.getId());
	}

}

@Service
public class StudentServiceImpl implements StudentService {

	@Autowired
	private StudentDao dao;

	@Override
	public Student read(Long id) {
		return dao.read(id);
	}

	@Override
	public List<Student> read() {
		return dao.read();
	}

	@Override
	public List<Student> read(Group group) {
		return dao.read(group);
	}

	@Override
	public void save(Student entity) {
		dao.save(entity);
	}
```

```
	@Override
	public void delete(Student entity) {
		dao.delete(entity.getId());
	}
}
```

9. Создать пакет runner и класс Main, предназначенный для запуска приложения.

```
@SpringBootApplication
@ComponentScan("lab2_2")
@Component
public class Main {
	public static void main(String[] args) {
		ConfigurableApplicationContext context = SpringApplica-
tion.run(Main.class);
		StudentService studentService = context.getBean(StudentService.class);
		studentService.save(createStudent());
		System.out.println(studentService.read());
		context.close();
	}

	private static Student createStudent() {
		Student student = new Student();
		student.setId(1L);
		student.setName("name");
		student.setSurname("surname");
		return student;
	}
}
```

10. Создать файл application.yml.

11. Запустить приложение и проверить работоспособность сервисов.

Архитектура данного приложения является трехслойной архитектурой сервера (dao, service, view). Однако в нашем случае отсутствует слой view, отвечающий за отображение и обработку действий пользователя. Каждый слой является независимым и для работы требует только внедрения компонентов низлежащего слоя.

Задание к лабораторной работе 4

Разработать логику класса GroupMemDao.

Лабораторная работа 5
Разработка приложения с использованием MVC и ORM

MVC (Model-view-controller или модель-представление-контроллер) – это архитектура проектирования, помогающая отделить представление от бизнес-логики.

Модель реализует основные операции доступа и управления данными и бизнес-логику, возможность регистрировать зависимые от него обработчики и

представления (наблюдателей). При изменениях – все зарегистрированные компоненты.

Представление представляет данные в некотором виде, читая их из модели при инициализации и после сообщений о произошедших изменениях. Кроме того, представление инициализирует связанный с ним контроллер.

Контроллер обрабатывает действия пользователя, транслируя их в операции над моделью или показ представлений.

Основные требования при разработке:

1. Информация может быть представлена по-разному и в нескольких местах для разных пользователей.

2. Изменения в данных должны немедленно отображаться в различных представлениях.

3. Простота изменения интерфейса, даже прямо во время работы приложения.

4. Перенос интерфейса между платформами не должен влиять на код, связанный с методами работы с данными и структурой данных приложения.

Основные шаги реализации:

– Выделить структуру данных, с которыми система работает, и набор операций над ними.

– Реализовать механизм передачи изменений.

– Реализовать необходимые представления.

– Реализовать необходимые обработчики действий пользователя.

– Спроектировать и реализовать связь между обработчиком и представлением.

– Реализовать построение системы из компонентов и их инициализацию.

ORM (Object-relational mapping или объектно-реляционное связывание) – принцип реализации связи с системами управления реляционными базами данных (СУБД) в Spring Framework. Доступ к данным реализуется при помощи шаблона проектирования Data access object (DAO). Устоявшегося русскоязычного эквивалента этому термину нет, его можно перевести как «Объектный доступ к данным». DAO – это объект, который предоставляет интерфейс к какому-либо типу базы данных или механизму хранения.

Методика выполнения задания

1. Создать базу данных в соответствии с предметной областью.

2. Создать новый Maven проект.

3. Сконфигурировать pom.xml для работы с Spring Data: …

```
<dependency>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
	<groupId>mysql</groupId>
	<artifactId>mysql-connector-java</artifactId>
	<version>5.1.47</version>
</dependency>
...
```

4. Создать сущности с маппингом для Hibernate:

a. Создать пакет для сущностей: lab4.entity.

b. Создать сущности.

Абстрактный суперкласс для всех сущностей должен содержать id и аннотацию `@MappedSuperclass`.

```
@MappedSuperclass
public abstract class AbstractEntity {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	protected Long id;

	//методы геттер, сеттер, equals, хэш-код
}
```

Далее создать классы-сущности:

группа:

```
@Entity
@Table(name = "groups")
@AttributeOverride(name = "id", column = @Column(name = "group_id"))
public class Group extends AbstractEntity {
	@Column(unique = true)
	private String grName;
	@OneToMany(mappedBy = "group", cascade = {CascadeType.ALL}, orphanRemoval = true)
	private Set<Student> students;

	// методы геттеры, сеттеры, toString
}
```

студент:

```
package lab4.entity;

@Entity
@Table(name = "students")
public class Student extends AbstractEntity {
	@Column(name = "name")
	private String name;
	@Column(name = "surname")
	private String surname;
	@Column(name = "phone_number")
	private String phoneNumber;
	@ManyToOne(cascade = {CascadeType.PERSIST, CascadeType.MERGE})
	@JoinColumn(name = "group_id", nullable = false)
	private Group group;
	// геттеры, сеттеры, toString
}
```

Примечание: поля, названия которых в базе данных идентичны названию полей в классе, могут не помечаться аннотацией `@Column`, однако, если в БД название поля отличается, то оно указывается в качестве параметра **name** аннотации `@Column`.

5. Создать respositories**.**

a. Создать пакет для объектов доступа к базе данных lab4.repository.

b. Создать repository для каждой сущности:

```
@Repository
public interface StudentRepository extends JpaRepository<Student, Long> {
	List<Student> findByGroupId(Long id);
	List<Student> findBySurname(String name);
}

@Repository
public interface GroupRepository extends JpaRepository<Group, Long> {
	Group findByGrName(String name);
}
```

6. Создать в папке resources файл с настройками spring boot и базы данных application.yml

```
 server:     port: 8088
 spring:   datasource:
    platform: MySQL
    driver-class-name: com.mysql.jdbc.Driver
    url:
jdbc:mysql://localhost:3306/university?useSSL=false&serverTimezone=UTC&useLegacyDatet
imeCode=false
    username: root
    password: 1234
  # JPA properties
  jpa:
    hibernate:
      ddl-auto: update # When you launch the application for the first time - switch
"none" at "create"
    show-sql: false     database: mysql     database-platform:
org.hibernate.dialect.MySQL55Dialect
    #  open-in-view: false generate-ddl: true
```

7. Создать сервисы и их реализации в соответствии с сущностями.

a. Создать пакет с интерфейсами сервисов: lab4.service.

b. Создать сервисы:

Абстрактный сервис:

```
public interface Service<T extends AbstractEntity> {
	T read(Long id);
	List<T> read();
	void save(T entity);
	void delete(Long id);
	void edit (T entity);
}

public interface StudentService extends Service<Student> {
	List<Student> readByGroupId(Long groupId);
	List<Student> readBySurname(String surname);
}

public interface GroupService extends Service<Group> {
	Group readByName(String name);
}
```

c. Создать реализации сервисов в пакете lab4.service.impl.

8. Создать главный класс программы**.**

a. Создать пакет для главного класса lab4.runner.

b. Создать точку входа в программу (класс с методом main):

```
@SpringBootApplication
@EnableJpaRepositories("lab4.repository")
@ComponentScan("lab4.service")
@EntityScan("lab4.entity")
public class Main {
        ConfigurableApplicationContext ctx = SpringApplication.run(Main.class);
        GroupService groupService = ctx.getBean(GroupService.class);
        StudentService studentService = ctx.getBean(StudentService.class);
        ...
        // вызов CRUD-методов сервисов
        ...
        ctx.close();
    }

}
```

9. Добавить данные в базу и запустить.

Задание к лабораторной работе 5:

a. В главном классе приложения сделать CRUD-запросы к таблице «students».

b. Добавить к проекту вертикаль (entity, repository, service):

Варианты 1,4,7,10,13: Факультет, в котором имеется название, телефон, список групп. Факультет д. б. связан с таблицей «группы» связью «1 ко многим».

Варианты 2,5,8,11,14: Специальность, которая содержит список групп, название и код специальности. Специальность д. б. связана с таблицей «группы» связью «1 ко многим».

Варианты 3,6,9,12,15: Ведущий факультатива, преподающий у студентов, у которого есть должность, имя, фамилия и телефон. Ведущий д. б. связан с таблицей «студенты» связью «1 ко многим».

В базе данных создать и заполнить соответствующие таблицы.

Для сдачи лабораторной требуется продемонстрировать работу методом CRUD в сервисах (create, read, update, delete). Методов read может быть несколько.

Лабораторная работа 6
Разработка приложения с использованием технологии REST

В технологии передачи репрезентативного состояния (Representational State Transfer – REST) используется протокол HTTP. Основная идея – назначение уникального идентификатора URL каждой веб-службе, каждому ресурсу.

Ресурс – это любая сущность, на которую клиент может поставить ссылку или с которой может попытаться взаимодействовать. Каждый ресурс должен иметь URI. Параметры веб-службе передаются как параметры форм. Каждая веб-служба отображается в один из методов протокола HTTP.

Представление ресурса – это любая полезная информация о состоянии ресурса.

Ресурсы должны быть связаны максимально сильно друг с другом (как вершины графов).

Основные принципы проектирования REST

REST предлагает разработчикам использовать HTTP-методы явно в соответствии с определением протокола. Этот основной принцип проектирования REST устанавливает однозначное соответствие между операциями create, read, update и delete (CRUD) и HTTP-методами. Согласно этому соответствию:

– для создания ресурса на сервере используется POST;
– для извлечения ресурса используется GET;
– для изменения состояния ресурса или его обновления используется PUT;
– для удаления ресурса используется DELETE.

Каждый HTTP-запрос происходит в изоляции от других, так как серверу никогда не приходится отслеживать запросы, выполненные ранее.

Методика выполнения задания

1. Создать проект maven.

2. Скопировать maven-зависимости из лабораторной работы 5.

3. Скопировать пакеты сущностей, repository, сервисов, а также содержимое папки resources из лабораторной работы 5.

4. Добавить в pom.xml зависимости для работы с Jackson:

```
…
<dependency>
	<groupId>com.fasterxml.jackson.core</groupId>
	<artifactId>jackson-core</artifactId>
</dependency>

<dependency>
	<groupId>com.fasterxml.jackson.core</groupId>
	<artifactId>jackson-databind</artifactId>
</dependency>
…
```

5. В классе студента добавить к полю group аннотацию `@JsonProperty(access = Access.WRITE_ONLY)`:

```
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "group_id", nullable = false)
@JsonProperty(access = Access.WRITE_ONLY)
private Group group;
public String getName() {
	return name;
}
```

6. Создать пакет controller и внутри него следующие классы:

абстрактный конроллер:

```
public abstract class AbstractController<T extends AbstractEntity> {
	protected HttpHeaders headers;

	@PostConstruct
	private void init() {
		headers = new HttpHeaders();
		headers.setContentType(MediaType.APPLICATION_JSON);
	}

	@GetMapping
	public ResponseEntity<List<T>> get() {
		List<T> entities = getService().read();
		if (entities.isEmpty()) {
			return new ResponseEntity<>(HttpStatus.NOT_FOUND);
		}
		return new ResponseEntity<>(entities, headers, HttpStatus.OK);
	}

	@GetMapping("/{id}")
	public ResponseEntity<T> getById(@PathVariable long id) {
		T entity = getService().read(id);
		if (entity == null) {
			return new ResponseEntity<>(HttpStatus.NOT_FOUND);
		}
		return new ResponseEntity<>(entity, headers, HttpStatus.OK);
	}

	@PutMapping
	public ResponseEntity<String> put(@RequestBody T entity) {
		getService().save(entity);
		return new ResponseEntity<>(HttpStatus.CREATED);
	}

	@PostMapping
	public ResponseEntity<String> post(@RequestBody T entity) {
		getService().save(entity);
		return new ResponseEntity<>(HttpStatus.OK);
	}

	@DeleteMapping("/{id}")
	public ResponseEntity<String> delete(@PathVariable long id) {
		getService().delete(id);
		return new ResponseEntity<>(HttpStatus.OK);
	}

	public abstract Service<T> getService();
}
```

контроллер группы:

```
@RestController
@RequestMapping("api/group")
public class GroupController extends AbstractController<Group> {

	@Autowired
	private GroupService service;

	@Override
```

```
	public GroupService getService() {
		return service;
	}

	@GetMapping("/name/{name}")
	public ResponseEntity<Group> getStudentsBySurname(@PathVariable String name) {
		Group group = service.readByName(name);
		if (group == null) {
			return new ResponseEntity<>(HttpStatus.NOT_FOUND);
		}
		return new ResponseEntity<>(group, headers, HttpStatus.OK);
	}

}
```

контроллер студента:

```
@RestController
@RequestMapping("api/student")
public class StudentController extends AbstractController<Student> {

	@Autowired
	private StudentService service;

	@Override
	public StudentService getService() {
		return service;
	}

	@GetMapping("/group/{id}")
	public ResponseEntity<List<Student>> getStudentsByGroup(@PathVariable long id) {
		List<Student> students = service.readByGroupId(id);
		if (students.isEmpty()) {
			return new ResponseEntity<>(HttpStatus.NOT_FOUND);
		}
		return new ResponseEntity<>(students, headers, HttpStatus.OK);
	}

	@GetMapping("/surname/{surname}")
	public ResponseEntity<List<Student>> getStudentsBySurname(@PathVariable String
	surname) {
		List<Student> students = service.readBySurname(surname);
		if (students.isEmpty()) {
			return new ResponseEntity<>(HttpStatus.NOT_FOUND);
		}
		return new ResponseEntity<>(students, headers, HttpStatus.OK);
	}
}
```

7. Создать главный класс приложения:

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Main {
	public static void main(String[] args) {
		SpringApplication.run(Main.class);
	}
}
```

Приложение будет доступно по адресу localhost:8088. Для тестирования нужно установить плагин Restfull client в браузере Mozilla firefox. URL плагина: https://addons.mozilla.org/en-US/firefox/addon/restclient/.

Задание к лабораторной работе 6

Добавить контроллер в соответствии с сущностью, созданной в лабораторной работе 5. Для сдачи лабораторной требуется продемонстрировать работу методов CRUD в веб-сервисе (create, read, update, delete).

Лабораторная работа 7
Разработка приложения с использованием фреймворка spring sequrity

Spring Security – это Java/JavaEE framework, предоставляющий механизмы построения систем аутентификации и авторизации, а также другие возможности обеспечения безопасности для корпоративных приложений, созданных с помощью Spring Framework. Механизм разграничения доступа добавляется к контроллерам при помощи аннотаций, либо при помощи xml-конфигурации.

Методика выполнения задания

За основу взять лабораторную работу 6.

1. Добавить к pom.xml зависимость Spring Security:

```
…
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
…
```

2. Создать класс WebSecurityConfig.

```
package lab4.config;

@Configuration
@EnableGlobalMethodSecurity(prePostEnabled = true)
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

	@Override
	protected void configure(HttpSecurity http) {
		http.httpBasic().and().authorizeRequests().antMatchers("/api/group/**",
		"/api/student/**").permitAll().and().csrf().disable();
	}
}
```

Для разграничения доступа пользователей соответствующие методы контроллеров требуется пометить аннотацией @PreAuthorise: `@PreAuthorize("hasRole('ADMIN')")` – роль администратора,

`@PreAuthorize("hasRole('USER')")` – роль пользователя.

3. Добавить в базу данных таблицы с ролями:

```
CREATE TABLE `users` (
	`id` INT NOT NULL AUTO_INCREMENT,
	`username` varchar(64) NOT NULL,
	`password` varchar(64) NOT NULL,
	`authority` varchar(64) NOT NULL,
	PRIMARY KEY (`id`)
);

insert into `users` (id, username, password, authority) values
	(1, 'admin', '$2a$10$jrryFNptnoGWwyWhxc47eeeHpin/LPOut7J221Xv4DB3qTswVcvJS',
'ROLE_ADMIN'),
	(2, 'user',
'$2a$10$I0BOCCDqRH6905RIlUmgd.2L008fmT3QvFtjEynyJQ2WoKDFRNGo6','ROLE_USER');
```

Здесь в графе «пароль» закодированные при помощи класса BCryptPasswordEncoder пароли. Для админа – admin, для юзера – user.

4. Создать сущность User:

```
@Entity
@Table(name = "users")
public class User extends AbstractEntity {
    @Column(nullable = false, unique = true)
    private String username;
    @Column(nullable = false)
    private String password;
    @Column(nullable = false)
    private String authority;

    // конструктор, геттеры, сеттеры
}
```

5. Создать Repository для User:

```
public interface UserRepository extends JpaRepository<User, Long> {
    User findByUsername(String username);
}
```

6. Создать класс CustomUserDetailService для авторизации при помощи базы данных:

```
@Service
public class CustomUserDetailService implements UserDetailsService {
	@Autowired
	private UserRepository repository;

	@Override
	public UserDetails loadUserByUsername(String username) throws UsernameNotFoundEx-
	ception {
		by.vstu.isap.zamok.lab5.entity.User user = repository
		.findByUsername(username);
		if (user == null) {
		throw new UsernameNotFoundException(username);
	  }
```

```
        List<SimpleGrantedAuthority> grantedAuthorities = new ArrayList<>();
          grantedAuthorities.add(new SimpleGrantedAuthority(user.getAuthority()));
          return new User(user.getUsername(), user.getPassword(), grantedAuthorities);
        }
}
```

7. Добавить сервис связи с БД в WebSecurityConfig:

`@Autowired private CustomUserDetailService service` и бины для инициализации связи:

```
    @Bean
    public DaoAuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider auth = new DaoAuthenticationProvider();
        auth.setUserDetailsService(service);
        auth.setPasswordEncoder(passwordEncoder());
        return auth;
    }
    @Override
    protected void configure(AuthenticationManagerBuilder auth) {
        auth.authenticationProvider(authenticationProvider());
    }
```

8. Проверка security при помощи Postman.

Запросы, не помеченные аннотацией @PreAuthorise, допускаются фильтром и не требуют аутентификации. При обращении к методу, помеченному аннотацией @PreAuthorise с параметром hasRole или hasAnyRole, требуется передача имени пользователя и пароля в каждом запросе. В случае если имя и пароль не передать, сервер вернет код ошибки 401. Если же пользователь не входит в группу разрешенных, то ответом будет код ошибки 403. При совпадении имени пользователя, пароля и требуемой роли запрос будет выполнен.

Задание к лабораторной работе 7

Создать REST сервер при помощи springframework. В проекте должна быть реализована трехслойная архитектура и взаимодействие с базой данных (БД). К проекту должен быть приложен скрипт создания БД.

№ варианта	Задание
1,6,11	**Информационная система библиотеки.** Информационная система библиотеки позволяет искать книги в своем каталоге [guest], учитывать выдачу книг на абонемент и возврат книг [manager], а также позволяет добавлять книги в фонд и списывать их [admin]
2,7,12	**Информационная система поликлиники.** Информационная система поликлиники позволяет ставить и снимать больных с учета [manager], записывать больных на прием к врачам [manager], учитывать факт приема [manager], позволяет вести историю болезни (медицинскую карту) больного [manager], добавлять и удалять врачей [admin] и просматривать список врачей [guest]
3,8,13	**Информационная система деканата.** Информационная система деканата позволяет принимать и отчислять студентов[manager], вести учет успеваемости по итогам сессии [manager], переводить студентов из группы в группу и с курса на курс [admin], просматривать студентов группы [guest]

4,9,14	**Система учета рабочего времени.** Система учета рабочего времени позволяет руководителям выдавать задания и отслеживать ход их выполнения [admin], а исполнителям – вести учет рабочего времени [manager], затраченного на выполнение каждого задания, а также узнать выданные сотрудникам задания [guest]
5,10,15	**Информационная система технической экспертизы.** Информационная система технической экспертизы позволяет соискателям грантов подавать заявки [guest], независимым экспертам оценивать заявки [manager], а держателям фонда принимать решение о выдаче грантов по результатам экспертизы заявок [admin]

В квадратных скобках указаны роли, которые имеют доступ к указанным возможностям бизнес-логики:

– guest – любой пользователь, в том числе неаутентифицированный;

– manager – пользователь с ограниченными правами использования системы;

– ad min – администратор системы с полными правами.

Методы CRUD (добавления, удаления, редактирования и просмотра сущностей) должны быть доступны только администратору, если условием не указано иное.

Все взаимодействие с сервером должно производиться при помощи REST-протокола.

Список рекомендуемой литературы

1. Дейтел, Х. М. Как программировать на С / Х. М. Дейтел, П. Дж. Дейтел. – М.: Бином, 2006. – 1037 с.

2. Страуструп, Б. Язык программирования С++, спец. изд. / Б. Страуструп. – М., СПб.: «Издательство БИНОМ» – «Невский диалект», 2001. – 1099 с.

3. Йодан, Э. Структурное программирование и конструирование программ / Э. Йодан. – М.: Мир, 1979.

4. Кормен, Т. Алгоритмы: Построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест. – СПб., 2003.

5. Бусько, В. Л. Основы алгоритмизации и программирования в среде Visual C++ : лабораторный практикум по курсу «Основы алгоритмизации и программирования» / В. Л. Бусько, А. А. Навроцкий. – Минск : БГУИР, 2008. – 66 с.

Учебное издание

КОМПОНЕНТНОЕ ПРОГРАММИРОВАНИЕ

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Составители:

Казаков Вадим Евгеньевич
Деркаченко Павел Григорьевич

Редактор *Т.А. Осипова*
Корректор *Т.А. Осипова*
Компьютерная верстка *П.Г. Деркаченко*

Подписано к печати 10.06.2021. Формат 60x90 $^{1}/_{16}$. Усл. печ. листов 2,1.
Уч.-изд. листов 2,7. Тираж 35 экз. Заказ № 130.

Учреждение образования «Витебский государственный технологический университет»
210038, г. Витебск, Московский пр., 72.
Отпечатано на ризографе учреждения образования
«Витебский государственный технологический университет».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий № 1/172 от 12 февраля 2014 г.
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий № 3/1497 от 30 мая 2017 г.