Лабораторная работа 5

Построение многослойной архитектуры приложения

I. Теоретическая часть

Архитектура приложений функционирует как план потока данных и информации, который может эффективно решать задачи. Продуманные функции и интерфейсы обеспечивают гибкость и расширяемость кода.

Система делится на уровни, каждый из которых взаимодействует лишь с двумя соседними. Поэтому запросы к БД, которая обычно располагается в самом конце цепочки взаимодействия, проходят последовательно сквозь каждый «слой».

Каждый уровень этой архитектуры выполняет строго ограниченный набор функций (которые не повторяются от слоя к слою) и не знает о том, как устроены остальные уровни. Поэтому «содержимое» уровней можно изменять без риска глобальных конфликтов между слоями.

Все сущности оформлены в виде сущностей (**Model**).

Они содержат набор атрибутов (приватные поля класса), конструкторы, сеттеры и геттеры для установки/чтения атрибутов. Иного кода в них нет. Часто подобные объекты называют POJO (Plain Old Java Object).

Всю логику работы с сущностями реализует слой **Service**.

Он формирует бизнес-правила для моделей. Среди аргументов запросов и возвращаемых результатов часто выступают сами сущности (или их коллекции).

Слой доступа к данным **Data Access Object (DAO)** — «посредник» между СУБД и Service, работающий непосредственно с базой данных, и отвечающий за взаимодействие с ней.

Каждый слой максимально изолирован от других. Если вместо БД будет набор текстовых файлов, то достаточно поменять только реализацию DAO, не трогая остальной код. Можно подключить другой Service с минимальными изменениями.

II. Практическая часть.

Рассмотрим построение архитектуры на простом примере, работающим со студентами.

Для описания сущности, создадим класс Student.

```
public class Student {
    private String firstName;
    private String lastName;
    private int age;
}
```

Для описания слоя доступа к данным, подготовим соответствующий интерфейс:

```
public interface Dao<T, ID> {
    T findById(ID id);
    Collection<T> findAll();
    T save(T entity);
    T update(T entity);
    void delete(T entity);
    void deleteById(ID id);
}
```

Он должен содержать базовые CRUD-методы, а также вспомогательные для выполнения соответствующих запросов.

Например, метод `findAll()` будет возвращать коллекцию. Так как это интерфейс, то в реализации необходимо будет возвращать конкретный тип.

В результате у нас получился компонент, реализующий логику доступа к данным, не имеющий зависимостей от сторонних библиотек.

Создадим класс, реализующего интерфейс.

```
public class StudentDao implements Dao<Student, Long>{ …
```

Например, рассмотрим реализацию получения списка студентов.

```
private final static String FIND_ALL = "SELECT * FROM students";
@Override
public List<Student> findAll() {
    List<Student> list = null;
    ResultSet rs = null;
    try(PreparedStatement statement =
           MainApplication
                .getConnection()
                .prepareStatement(FIND_ALL)) {
       rs = statement.executeQuery();
       list = mapper(rs);
```

```
    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
  return list;
}
```

Подключение к БД и получение единственного Connection можно вынести в утилитный класс. Само соединение можно устанавливать при старте программы и закрывать при ее завершении.

```
private static Connection connection;
public void start(Stage stage) throws IOException {
try {
       connection = DBHelper.getConnection();
   } catch (SQLException e) {
       System.out.println("Ошибка подключения");
       System.out.println(e.getMessage());
   }
...
}
@Override
public void stop() throws Exception {
   if(connection != null) {
       connection.close();
   }
   super.stop();
}
```

Также определен метод *mapper,* предназначенный для формирования экземпляра класса Student из ответа от СУБД в виде ResultSet.

```
protected List<Student> mapper(ResultSet rs){
   List<Student> list = new ArrayList<>();
   try {
       while (rs.next()) {
           list.add(new Student(rs.getString("firstname"),
                       rs.getString("lastname"), rs.getInt("age")));
       }
   } catch (SQLException e) {
       System.out.println(e.getMessage());
   }
```

```
   return list;
}
```

Работа с данными будет выполняться в классе контроллера

Для этого необходимо предусмотреть поле для работы с DAO, который инициализировать в конструкторе:

```
private StudentDao dao;
public MainController() {
   this.dao = new StudentDao();
}
```

В методе контроллера получаем этот список:

```
students.addAll(dao.findAll());
```

Далее с этим списком можно работать в соответствии с логикой программы и, например, отобразить в таблицу на форму.

Аналогично, можно реализовать добавление студента.

В контроллере получаем от пользователя или с формы нужные данные и передаем их в метод *save()*.

```
private final static String SAVE =
   "INSERT INTO students (firstname, lastname, age) VALUES (?, ?, ?)";
@Override
public Student save(Student student) {
    try(PreparedStatement statement =
            MainApplication.getConnection().prepareStatement(SAVE)){
       statement.setString(1, student.getName());
       statement.setString(2, student.getLastName());
       statement.setInt(3, student.getAge());
       statement.executeUpdate();
    } catch (SQLException e){
       System.out.println(e.getMessage());
    }
    return student;
}
```

Задание на лабораторную работу.

1. Создать базу данных INSTITUT

2. Добавить в БД таблицу STUDENTS, содержащую поля имя, фамилию, отчество, группа, возраст, город.

3. Реализовать в слое DAO методы

```
T findById(ID id);
Collection<T> findAll();
T save(T entity);
T update(T entity);
void delete(T entity);
void deleteById(ID id);
```

3. Реализовать в программе поиск студента по фамилии или группе, добавив в соответствующие методы в слой доступа.

4. Продемонстрировать в программе работу с этими методами. Данный выводить в табличном виде. Ввод новых данных и редактирование выполнять в отдельном окне. Информировать пользователя о результатах операций. Сообщения об ошибках выводить с использованием *Alert*.

Содержание отчета по лабораторной работе

- Титульный лист
- Описание задания
- Скриншоты разработанных экранных форм
- Разметка экранных форм на языке fxml
- Программный код основного класса, классов контроллеров, модели и DAO.

Вопросы для самоконтроля

1. Что такое компонент TableView?
2. Чем характеризуется контейнер AnchorPane?
3. Порядок определения *привязок* в AnchorPane?
4. Что такое многослойная архитектура приложения?
5. Для чего выделяется отдельный слой доступа к данным?
6. В чем отличие абстрактного класса от интерфейса?
7. Что такое параметризованный тип?

Список литературы

1. Вязовик, Н. А. Программирование на Java : учебное пособие / Н. А. Вязовик. — 3-е изд. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2021. — 601 c. — ISBN 978-5-4497-0852-6. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/102048.html

2. Мухаметзянов, Р. Р. Основы программирования на Java : учебное пособие / Р. Р. Мухаметзянов. — Набережные Челны : Набережночелнинский государственный педагогический университет, 2017. — 114 с. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/66812.html

3. Блох, Дж. Java. Эффективное программирование / Дж. Блох ; перевод В. Стрельцов ; под редакцией Р. Усманов. — 2-е изд. — Саратов : Профобразование, 2019. — 310 с. — ISBN 978-5-4488-0127-3. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/89870.html