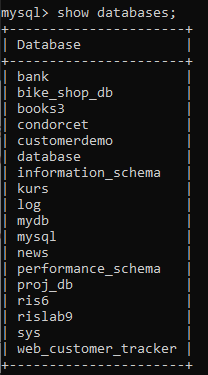
**Задание №4 JDBC**

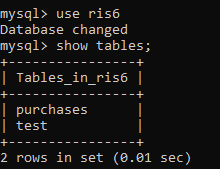
1. Проверьте наличие MySQL Server на вашем ПК.

Зайдите в командную строку MySQL Command Line Client.

В данной консоли вы сможете управлять СУБД MySQL. Далее мы с вами это сделаем на практике (Ps команды очень похожи на те, что вы использовали в MS SQL. Мы с вам будем использовать так же конструкторы БД, однако, знания скрипта вам нужны будут всегда, поэтому никогда их не забывайте):



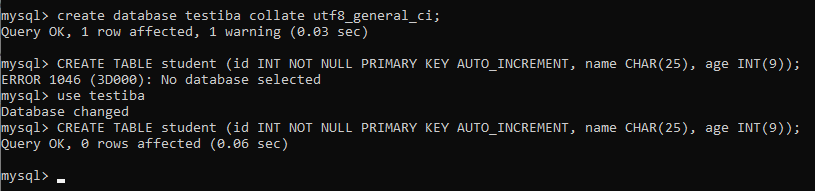
Вы так же можете выполнить следующее – выбрать существующую БД из списка с помощью команды USE. В моем случае я выбрал ris6, т.к. она у меня была, но вы сможете таким образом выбрать любую. Помните, что системный БД лучше не трогать, если не уверены зачем нужна та или иная информация в них:



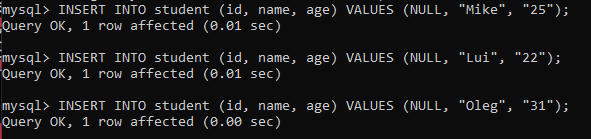
Далее, для проверки используем команды:

1. Создание БД.

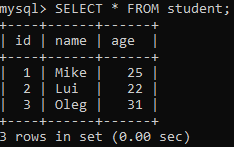
После строки создания выполните команду USE. А у же потом CREATE.



1. Добавим данные в БД

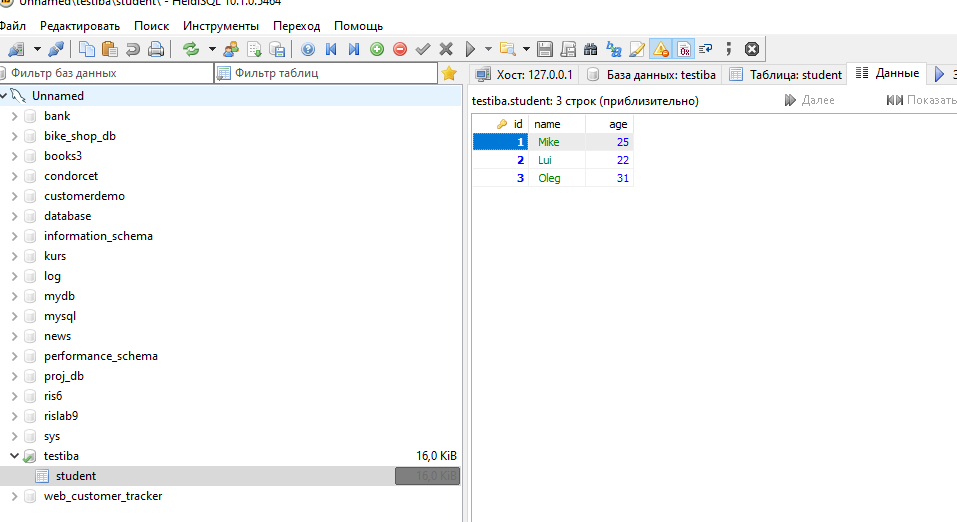


1. Произведем выборку из БД

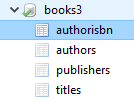


1. Перенесемся с вами в HediSQL.

Это простой способ подружится с БД без консоли и скрипта (ЛР 21.01):

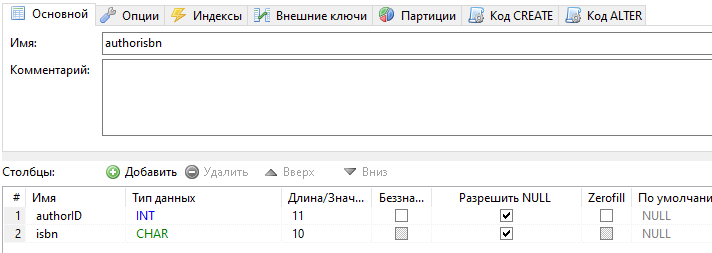


1. Создадим БД для нашего проекта:

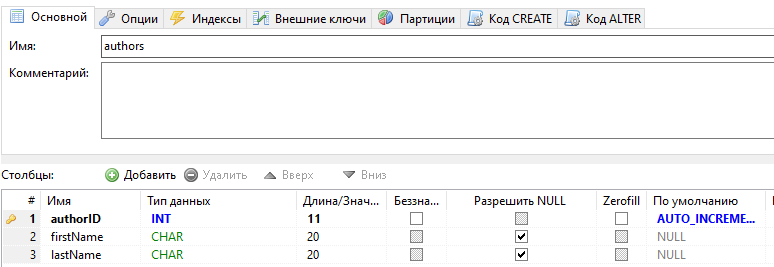


Каждая сущность будет содержать следующие поля (обратите внимание как создать первичный ключ):

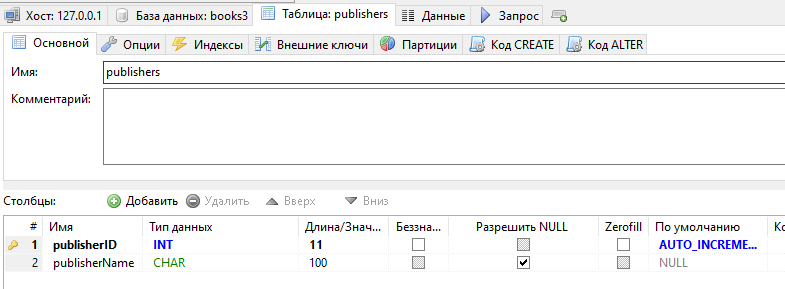
1. authorisbn



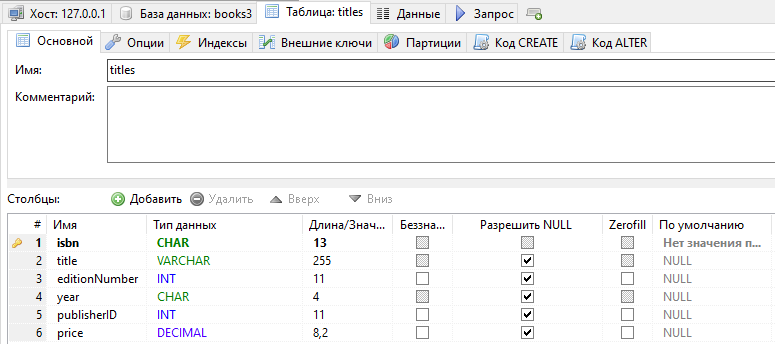
1. authors



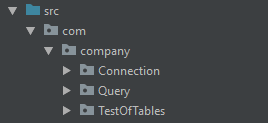
1. publishers



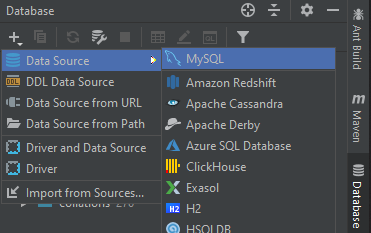
1. titles

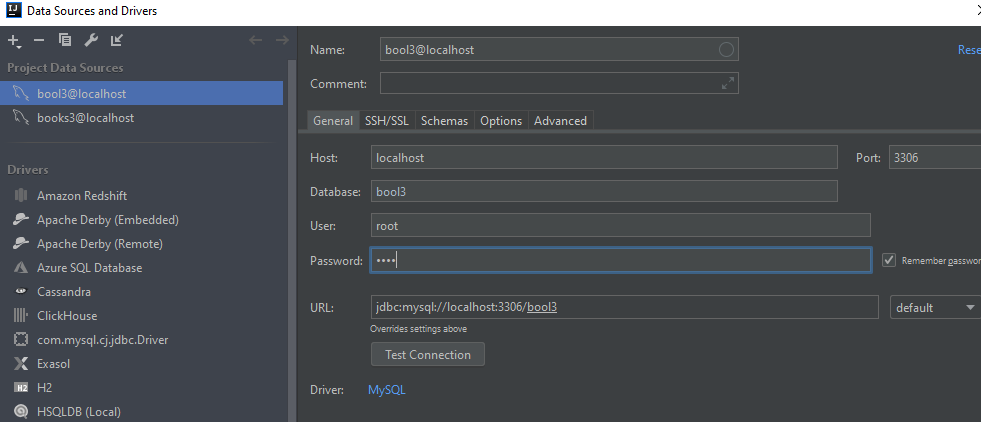


1. Создайте проект с помощью Maven.
2. Можно ориентироваться на следующую структуру нашего проекта:



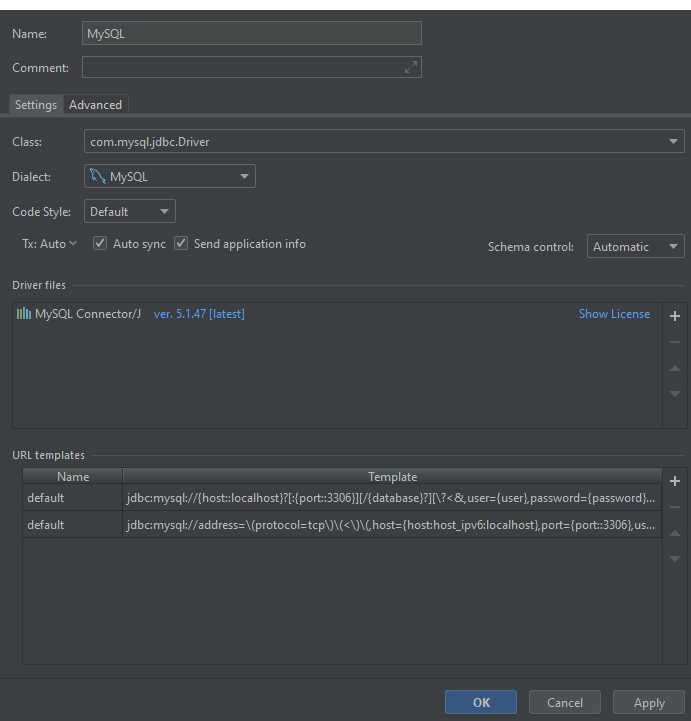
1. Для Ultimate версии можно подключить БД к проекту. Как производить настройку смотрите запись ЛР 21.01:





Далее можно настроить mysql-connector.

Нажимаем на MySQL и попадаем в настройки драйвера для БД.



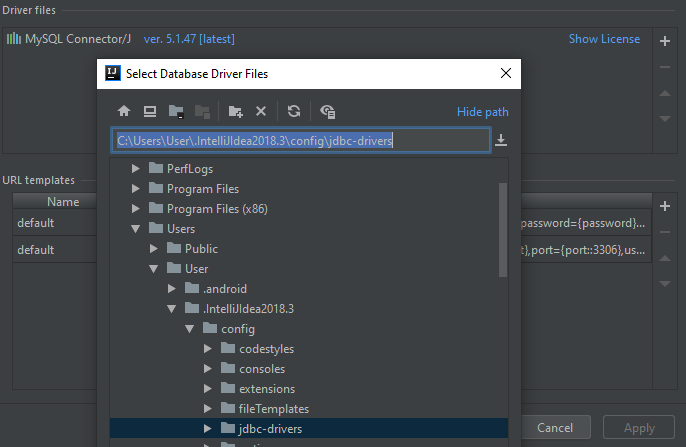
1. Начнем с подключения к БД.

В пакете Connection создайте 2 класса:



При попытки добавления драйвера, нам будет доступно выбрать путь к jar файлу коннектора.

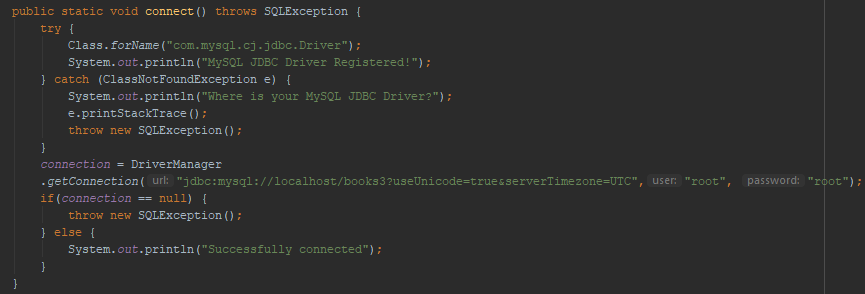
По умолчанию идет направление на встроенные драйвера Idea (J/Connector – урезанная версия mysql-connector)



1. Разберем содержимое и назначение данных классов Разбор содержимого классов смотрим в записи ЛР 21.01.

**Класс JDBC**

package com.company.Connection;  
  
import java.util.\*;  
import java.sql.\*;  
  
public class JDBC {  
  
 public static Connection *connection* = null;



public static void close() {  
 try {  
 if(*connection* != null) {  
 *connection*.close();  
 System.*out*.println("Closing connection");  
 }  
 } catch (SQLException e) {  
 System.*out*.println("Failed to close connection!");  
 }  
 }  
   
}

**Класс TestDatabase**

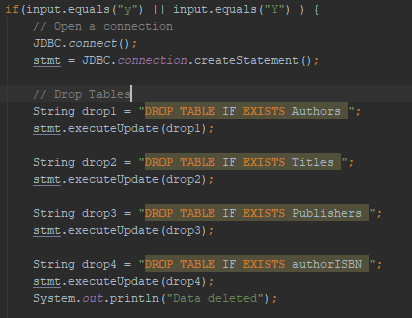
package com.company.Connection;  
  
import com.company.Connection.JDBC;  
import java.sql.\*;  
import java.util.\*;  
  
public class TestDatabase {  
  
 public static void main(String[] argv) {  
  
 System.*out*.println("-------- MySQL JDBC Connection Testing ------------");  
  
 try {  
 JDBC.*connect*();  
 } catch (SQLException e) {  
 System.*out*.println("Connection Failed!");  
 e.printStackTrace();  
 return;  
 }  
  
 JDBC.*close*();  
 }  
}

1. Далее создаем следующие классы:



**Класс CreateTables**

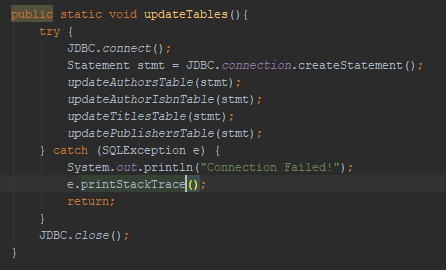
package com.company.TestOfTables;  
  
import com.company.Connection.JDBC;  
  
import java.sql.\*;  
import java.util.\*;  
  
public class CreateTables {  
  
 public static void main(String[] args) {  
 Statement stmt = null;  
 try{  
  
 System.*out*.println("This will DELETE all data, do you want to continue? (y/n) ");  
 Scanner scanner = new Scanner(System.*in*);  
 String input = scanner.nextLine();



// Create Tables  
 String authorsTable = "CREATE TABLE Authors " +  
 "(authorID **INTEGER** NOT NULL AUTO\_INCREMENT, " +  
 " firstName **CHAR**(20), " +  
 " lastName **CHAR**(20), " +  
 " PRIMARY KEY (authorID))";  
  
 stmt.executeUpdate(authorsTable);  
 System.*out*.println("Created Authors table");  
  
 String titlesTable = "CREATE TABLE Titles " +  
 "(isbn **CHAR**(13) not NULL, " +  
 " title **VARCHAR**(255), " +  
 " editionNumber **INTEGER**, " +  
 " year **CHAR**(4), " +  
 " publisherID **INTEGER** REFERENCES Publishers(publisherID), " +  
 " price **DECIMAL**(8,2), " +  
 " PRIMARY KEY (isbn))";  
  
 stmt.executeUpdate(titlesTable);  
 System.*out*.println("Created Titles table");  
  
 String publishersTable = "CREATE TABLE Publishers " +  
 "(publisherID **INTEGER** NOT NULL AUTO\_INCREMENT, " +  
 " publisherName **CHAR**(100), " +  
 " PRIMARY KEY (publisherID))";  
  
 stmt.executeUpdate(publishersTable);  
 System.*out*.println("Created Publishers table");  
  
 String authorISBNTable = "CREATE TABLE authorISBN " +  
 "(authorID **INTEGER** REFERENCES Authors(authorID), " +  
 " isbn **CHAR**(10) REFERENCES Titles(isbn))";  
  
 stmt.executeUpdate(authorISBNTable);  
 System.*out*.println("Created authorISBN table");  
  
 InsertTestData.*updateTables*();  
 }  
  
 } catch(SQLException se) {  
 // Handle errors for JDBC  
 se.printStackTrace();  
 } finally {  
 // Finally block, used to close resources  
 if(stmt != null) {  
 JDBC.*close*();  
 }  
 }  
 }  
}

**Класс InsertTestData**

package com.company.TestOfTables;  
  
import com.company.Connection.JDBC;  
  
import java.sql.\*;  
import java.util.\*;  
public class InsertTestData {  
  
 public static void main(String[] argv) {  
 *updateTables*();  
 }



private static void updateAuthorsTable(Statement stmt) {  
 String authorFirstNames[] = { "Jane", "Dan", "Ralph Waldo", "F.Scott","John", "Ernest", "Walter", "Stephen", "Stieg",  
 "George", "Joanne K.", "John", "John R. R.", "Kurt", "Andy" };  
 String authorLastNames[] = { "Austen", "Brown", "Emerson", "Firtzgerald", "Grisham", "Hemingway", "Isaacson",  
 "King", "Larsson", "Orwell", "Rowling", "Steinbeck", "Tolkien", "Vonnegut", "Weir" };  
 // reset the auto increment  
 String updateAuthorsTable = "ALTER TABLE Authors AUTO\_INCREMENT = 1";  
 try {  
 stmt.executeUpdate(updateAuthorsTable);  
 } catch (SQLException e) {  
 System.*out*.println("Execute Update Failed!");  
 e.printStackTrace();  
 return;  
 }  
 for (int i = 1; i <= authorFirstNames.length; i++) {  
 updateAuthorsTable = "INSERT INTO Authors (firstName, lastName)" + "VALUES ('" + authorFirstNames[i - 1]  
 + "', '" + authorLastNames[i - 1] + "');";  
 try {  
 stmt.executeUpdate(updateAuthorsTable);  
 } catch (SQLException e) {  
 System.*out*.println("Execute Update Failed!");  
 e.printStackTrace();  
 return;  
 }  
 }  
 }  
  
 private static void updateAuthorIsbnTable(Statement stmt) {  
 String isbn[] = { "0141439519", "0307474278", "0142437629", "0743273565", "0345543240", "0684801223",  
 "1501127625", "1501175466", "0307949486", "0451524935", "0439708180", "0142000687", "0547928227",  
 "0385333849", "0553418026" };  
 for (int i = 1; i <= isbn.length; i++) {  
 String updateAuthorISBNTable = "INSERT INTO authorISBN (authorID, isbn)" + "VALUES (" + i + ", '"  
 + isbn[i - 1] + "')";  
 try {  
 stmt.executeUpdate(updateAuthorISBNTable);  
 } catch (SQLException e) {  
 System.*out*.println("Execute Update Failed!");  
 e.printStackTrace();  
 return;  
 }  
 }  
 }  
  
 private static void updateTitlesTable(Statement stmt) {  
 String isbn[] = { "0141439519", "0307474278", "0142437629", "0743273565", "0345543240", "0684801223",  
 "1501127625", "1501175466", "0307949486", "0451524935", "0439708180", "0142000687", "0547928227",  
 "0385333849", "0553418026" };  
 String editionNumber[] = { "10", "10", "8", "4", "5", "7", "3", "14", "5", "3", "1", "18", "12", "2", "3" };  
 String year[] = { "2002", "2009", "2003", "2004", "2014", "1995", "2015", "2017", "2011", "1961", "1999",  
 "2002", "2012", "1999", "2014" };  
 String publisherID[] = { "1", "2", "1", "3", "4", "3", "5", "3", "6", "7", "8", "1", "9", "10", "11" };  
 String price[] = { "8.00", "8.99", "13.58", "6.99", "8.49", "9.98", "11.99", "9.55", "8.60", "7.64", "7.47",  
 "13.60", "11.16", "12.51", "10.76" };  
 String title[] = { "Pride and Prejudice", "The Da Vinci Code", "Nature and Selected Essays", "The Great Gatsby",  
 "Sycamore Row", "The Old Man and The Sea", "Steve Jobs", "It: A Novel",  
 "The Girl with the Dragon Tattoo", "1984", "Harry Potter and the Sorcerer''s Stone", "Cannery Row",  
 "The Hobbit", "Slaughterhouse-Five", "The Martian" };  
  
 for (int i = 0; i < isbn.length; i++) {  
 String updateTitlesTable = "INSERT INTO Titles (isbn, title, editionNumber, year, publisherID, price)"  
 + "VALUES ('" + isbn[i] + "', '" + title[i] + "', " + editionNumber[i] + ",'" + year[i] + "',"  
 + publisherID[i] + "," + price[i] + ")";  
 try {  
 stmt.executeUpdate(updateTitlesTable);  
 } catch (SQLException e) {  
 System.*out*.println("Execute Update Failed!");  
 e.printStackTrace();  
 return;  
 }  
 }  
 }  
  
 private static void updatePublishersTable(Statement stmt) {  
 String publishers[] = { "Penguin", "Anchor", "Scribner", "Dell Books", "Simon & Schuster", "Vintage Crime",  
 "Signet", "Scholastic", "Houghton Mifflin", "Dial Press", "Broadway" };  
 // reset the auto increment  
 String updatePublishersTable = "ALTER TABLE Publishers AUTO\_INCREMENT = 1";  
 try {  
 stmt.executeUpdate(updatePublishersTable);  
 } catch (SQLException e) {  
 System.*out*.println("Execute Update Failed!");  
 e.printStackTrace();  
 return;  
 }  
 for (int i = 0; i < publishers.length; i++) {  
 updatePublishersTable = "INSERT INTO Publishers (publisherName)" + "VALUES ('" + publishers[i] + "')";  
 try {  
 stmt.executeUpdate(updatePublishersTable);  
 } catch (SQLException e) {  
 System.*out*.println("Execute Update Failed!");  
 e.printStackTrace();  
 return;  
 }  
 }  
 }  
}

**Класс GetAllRows**

package com.company.TestOfTables;  
  
import com.company.Connection.JDBC;  
  
import java.sql.\*;  
import java.util.\*;  
  
public class GetAllRows {  
   
 public static void main(String[] args) {  
 Statement stmt = null;  
 try{  
  
 JDBC.*connect*();  
   
 stmt = JDBC.*connection*.createStatement();  
   
 String exampleQuery1 = "SELECT \* FROM Authors";  
 System.*out*.println("Authors:");  
 ResultSet rs1 = stmt.executeQuery(exampleQuery1);  
 while (rs1.next()) {  
 int id = rs1.getInt("authorID");  
 String firstName = rs1.getString("firstName");  
 String lastName = rs1.getString("lastName");  
 System.*out*.println(id + "\t" + firstName + "\t" + lastName);  
 }

**Запросы получения значений остальных сущностей нужно реализовать самостоятельно.**

**Внизу вы видите кусочек данного класса, не забываем добавить эту обработку ошибок.**

} catch(SQLException se) {  
 // Handle errors for JDBC  
 se.printStackTrace();  
 } finally {  
 // Finally block, used to close resources  
 JDBC.*close*();  
 }  
 }  
}

**Пример тестового задания на ДЗ:**

Необходимо сделать выборку данных всех издател. Для этого нам понадобится выполнить следующие действия:

1. Создадим класс (можете просто создавать методы и потом их вызывать из главного класса управления БД. Однако для начала, можете просто создавать классы для каждых запросов, в качестве обучения.)

package com.company.Query;  
import com.company.Connection.JDBC;  
  
import java.sql.\*;  
import java.util.\*;  
  
public class Query2 {  
  
 public static void main(String[] args) {  
 Statement stmt = null;  
 try{  
  
 JDBC.*connect*();  
  
 stmt = JDBC.*connection*.createStatement();

В приведенном классе мы сможем увидеть подключение библиотек. Объявление Statement – интерфейса, позволяющего выполнять SQL команды. Открытие коннекшина к БД с помощью метода connect.

И непосредственно инициализацию Statement.

Далее:

String query2Author = "SELECT \* FROM publishers";  
  
System.*out*.println("Show all publishers");  
  
ResultSet rs1 = stmt.executeQuery(query2Author);

Создадим новый запрос и поместим в строковую переменную. После чего передадим ее в метод выполнения запросов.

Далее:

while (rs1.next()) {  
 int id = rs1.getInt("publisherID");  
 String pubName = rs1.getString("publisherName");  
 System.*out*.println(id + "\t" + pubName);  
 }  
  
 } catch(SQLException se) {  
 //Handle errors for JDBC  
 se.printStackTrace();  
 } finally {  
 //finally block used to close resources  
 JDBC.*close*();  
 }  
 }

}

Пройдемся по результату выборки и закроем соединение.

**Задания для выполнения ЛР:**

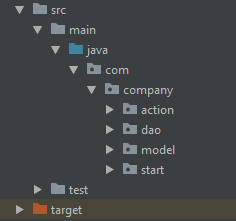
1. Сделайте выборку по авторам, отсортировав по их Имени и Фамилии
2. Добавьте нового Издателя (publusher).
3. Сделайте выборку Издателей и измените имя определенного Издателя.
4. Предоставьте отсортированный список книг определенного издателя (при этом id требуемого издателя можно менять в sql запросе)
5. Выполните добавление Нового автора в БД
6. Обновите Имя автора по определенному id
7. Добавить нового Publisher

Добавить новую Titles (При передачи VALUES publisherID – нужно сделать подзапросом select\*from publisher where publisherName =””)

Добавить authorISBN (при передачи VALUES необходимо параметр autorID так же сделать подзапросом с указанием имени и фамилии)

**PreparedStatement (Пояснения в записи 30.01)**

1. Создадим проект опираясь на следующую структуру:



Данный вид приближен к правильному распределению классов по задачам и разделению на слои приложения.

Отдельный пакет **dao** служит для подключения к БД и созданию запросов для получения или добавления данных.

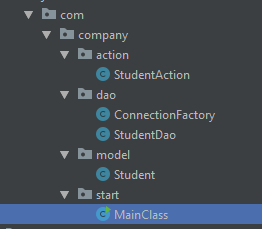
Пакет **model** необходим для представления объекта, которым мы будем манипулировать в процессе запросов к БД.

Пакет **start** содержит наш основной класс с точкой входа.

И наконец пакет **action** будет содержать бизнес-логику приложения, которая может выполнятся системой.

Сейчас с точки зрения работы с БД, нам будут в большей степени интересен пакет dao.

1. Создадим классы:



1. **MainClass**

package com.company.start;  
  
import com.company.action.StudentAction;  
import com.company.model.Student;  
  
import java.math.BigDecimal;  
import java.sql.Date;  
import java.util.Scanner;  
  
  
public class MainClass {  
  
 StudentAction action = new StudentAction();  
  
 public static void main(String[] args) {  
 String DbOperation = "EXIT(0), ADD(1), UPDATE(2), DELETE(3), FETCHBYID(4), FETCHBYEMAIL(5), FETCHBYMOBNO(6), FETCHBYNAME(7),"  
 + "FETCHBYCITY(8), FETCHBYSALRANGE(9), FETCHBYDOB(10), FETCHBYDOJRANGE(11), FETCHALL(12)";  
  
 MainClass mainclass = new MainClass();  
 int value = 0;  
 do {  
 System.*out*.println("valid operations are");  
  
 System.*out*.println(DbOperation);  
 System.*out*.println("Enter valid operation number 0-12");  
 Scanner scanner = new Scanner(System.*in*);  
 value = scanner.nextInt();  
  
 switch (value) {  
 case 1:  
 mainclass.addStudent();  
 break;  
 case 2:  
 mainclass.updateStudent();  
 mainclass.fetchAllStudent();  
 break;  
 case 3:  
 mainclass.deleteStudent();  
 break;  
 case 4:  
 mainclass.fetchStudentById();  
 break;  
 case 5:  
  
 break;  
 case 6:  
  
 break;  
 case 7:  
  
 break;  
 case 8:  
 mainclass.fetchStudentByCity();  
 break;  
 case 9:  
  
 break;  
 case 10:  
 mainclass.fetchStudentByDob();  
 break;  
 case 11:  
  
 break;  
 case 12:  
 mainclass.fetchAllStudent();  
 break;  
 case 0:  
 System.*out*.println("Exiting code");  
 break;  
 default:  
 System.*out*.println("Not a valid entry");  
 }  
 } while (value != 0);  
  
 }  
  
 public void addStudent() {  
 Student student = new Student();  
 Scanner insert = new Scanner(System.*in*);  
 System.*out*.println("Enter First Name");  
 student.setFname(insert.next());  
 System.*out*.println("Enter Last Name");  
 student.setLname(insert.next());  
 System.*out*.println("Enter Address");  
 student.setAddress(insert.next());  
 System.*out*.println("Enter Mobile Number");  
 student.setMobileNo(insert.next());  
 System.*out*.println("Enter Mail Id");  
 student.setMailId(insert.next());  
 System.*out*.println("Enter City");  
 student.setCity(insert.next());  
 System.*out*.println("Enter Designation");  
 student.setDesignation(insert.next());  
 System.*out*.println("Enter Dob (yyyy-mm-dd)");  
 student.setDob(Date.*valueOf*(insert.next()));  
 System.*out*.println("Enter Doj (yyyy-mm-dd)");  
 student.setDoj(Date.*valueOf*(insert.next()));  
 System.*out*.println("Enter Salary");  
 student.setSalary(insert.nextBigDecimal());  
 action.insert(student);  
 }  
  
 public void updateStudent() {  
 Student student = new Student();  
 Scanner insert = new Scanner(System.*in*);  
 System.*out*.println("Enter Student Id");  
 student.setId(insert.nextLong());  
 System.*out*.println("Enter First Name");  
 student.setFname(insert.next());  
 System.*out*.println("Enter Last Name");  
 student.setLname(insert.next());  
 System.*out*.println("Enter Address");  
 student.setAddress(insert.next());  
 System.*out*.println("Enter Mobile Number");  
 student.setMobileNo(insert.next());  
 System.*out*.println("Enter Mail Id");  
 student.setMailId(insert.next());  
 System.*out*.println("Enter City");  
 student.setCity(insert.next());  
 System.*out*.println("Enter Designation");  
 student.setDesignation(insert.next());  
 System.*out*.println("Enter Dob (yyyy-mm-dd)");  
 student.setDob(Date.*valueOf*(insert.next()));  
 System.*out*.println("Enter Doj (yyyy-mm-dd)");  
 student.setDoj(Date.*valueOf*(insert.next()));  
 System.*out*.println("Enter Salary");  
 student.setSalary(insert.nextBigDecimal());  
 action.update(student);  
 }  
  
 public void deleteStudent() {  
 Scanner insert = new Scanner(System.*in*);  
 System.*out*.println("Enter Student Id");  
 long id = insert.nextLong();  
 action.delete(id);  
 }  
  
 public void fetchStudentById() {  
 Scanner insert = new Scanner(System.*in*);  
 System.*out*.println("Enter Student Id");  
 long id = insert.nextLong();  
 action.fetchById(id);  
 }  
  
 public void fetchStudentByEmail() {  
 Scanner insert = new Scanner(System.*in*);  
 System.*out*.println("Enter Student Mail Id");  
 String mailId = insert.next();  
 action.fetchByEmailId(mailId);  
 }  
  
 public void fetchStudentByMobileNo() {  
 Scanner insert = new Scanner(System.*in*);  
 System.*out*.println("Enter Student Mobile Number");  
 String mobileNo = insert.next();  
 action.fetchByMobileNo(mobileNo);  
 }  
  
 public void searchStudentByName() {  
 Scanner insert = new Scanner(System.*in*);  
 System.*out*.println("Enter Student Name");  
 String name = insert.next();  
 action.searchByName(name);  
 }  
  
 public void fetchStudentByCity() {  
 Scanner insert = new Scanner(System.*in*);  
 System.*out*.println("Enter Student City");  
 String city = insert.next();  
 action.fetchByCity(city);  
 }  
  
 public void fetchStudentBySalaryRange() {  
 Scanner insert = new Scanner(System.*in*);  
 System.*out*.println("Enter Salary Start Range");  
 BigDecimal salaryRange1 = insert.nextBigDecimal();  
 System.*out*.println("Enter Salary End Range");  
 BigDecimal salaryRange2 = insert.nextBigDecimal();  
  
 action.fetchBySalaryRange(salaryRange1, salaryRange2);  
 }  
  
 public void fetchStudentByDob() {  
 Scanner insert = new Scanner(System.*in*);  
 System.*out*.println("Enter Date of Birth (yyyy-mm-dd)");  
 String dob = insert.next();  
 action.fetchByDob(Date.*valueOf*(dob));  
 }  
  
 public void fetchStudentByDOjRange() {  
 Scanner insert = new Scanner(System.*in*);  
 System.*out*.println("Enter Start Date of Joining (yyyy-mm-dd)");  
 String dob1 = insert.next();  
 System.*out*.println("Enter End Date of Joining (yyyy-mm-dd)");  
 String dob2 = insert.next();  
 action.fetchByRangeDoj(Date.*valueOf*(dob1), Date.*valueOf*(dob2));  
 }  
  
 public void fetchAllStudent() {  
 action.fetchAll();  
 }  
  
}

Класс **Student**

package com.company.model;  
  
import java.math.BigDecimal;  
import java.sql.Date;  
import java.sql.Timestamp;  
  
public class Student {  
  
 private long id;  
 private String fname;  
 private String lname;  
 private String address;  
 private String mobileNo;  
 private String mailId;  
 private String city;  
 private String designation;  
 private Date dob;  
 private Date doj;  
 private BigDecimal salary;  
 private Timestamp addDate;  
  
 public Student() {  
 }  
  
 public Student(String fname, String lname, String address, String mobileNo, String mailId, String city, String designation, Date dob, Date doj, BigDecimal salary) {  
 this.fname = fname;  
 this.lname = lname;  
 this.address = address;  
 this.mobileNo = mobileNo;  
 this.mailId = mailId;  
 this.city = city;  
 this.designation = designation;  
 this.dob = dob;  
 this.doj = doj;  
 this.salary = salary;  
 }  
  
 public Student(long id, String fname, String lname, String address, String mobileNo, String mailId, String city, String designation, Date dob, Date doj, BigDecimal salary) {  
 this.id = id;  
 this.fname = fname;  
 this.lname = lname;  
 this.address = address;  
 this.mobileNo = mobileNo;  
 this.mailId = mailId;  
 this.city = city;  
 this.designation = designation;  
 this.dob = dob;  
 this.doj = doj;  
 this.salary = salary;  
 }  
  
 @Override  
 public String toString() {  
 return "Student{" + "id=" + id + ", fname=" + fname + ", lname=" + lname + ", address=" + address + ", mobileNo=" + mobileNo + ", mailId=" + mailId + ", city=" + city + ", designation=" + designation + ", dob=" + dob + ", doj=" + doj + ", salary=" + salary + ", addDate=" + addDate + '}';  
 }  
  
 public long getId() {  
 return id;  
 }  
  
 public void setId(long id) {  
 this.id = id;  
 }  
  
 public String getFname() {  
 return fname;  
 }  
  
 public void setFname(String fname) {  
 this.fname = fname;  
 }  
  
 public String getLname() {  
 return lname;  
 }  
  
 public void setLname(String lname) {  
 this.lname = lname;  
 }  
  
 public String getAddress() {  
 return address;  
 }  
  
 public void setAddress(String address) {  
 this.address = address;  
 }  
  
 public String getMobileNo() {  
 return mobileNo;  
 }  
  
 public void setMobileNo(String mobileNo) {  
 this.mobileNo = mobileNo;  
 }  
  
 public String getMailId() {  
 return mailId;  
 }  
  
 public void setMailId(String mailId) {  
 this.mailId = mailId;  
 }  
  
 public String getCity() {  
 return city;  
 }  
  
 public void setCity(String city) {  
 this.city = city;  
 }  
  
 public Timestamp getAddDate() {  
 return addDate;  
 }  
  
 public void setAddDate(Timestamp addDate) {  
 this.addDate = addDate;  
 }  
  
 public String getDesignation() {  
 return designation;  
 }  
  
 public void setDesignation(String designation) {  
 this.designation = designation;  
 }  
  
 public Date getDob() {  
 return dob;  
 }  
  
 public void setDob(Date dob) {  
 this.dob = dob;  
 }  
  
 public Date getDoj() {  
 return doj;  
 }  
  
 public void setDoj(Date doj) {  
 this.doj = doj;  
 }  
  
 public BigDecimal getSalary() {  
 return salary;  
 }  
  
 public void setSalary(BigDecimal salary) {  
 this.salary = salary;  
 }  
}

Класс **StudentDao**

package com.company.dao;  
  
import java.sql.Connection;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.util.ArrayList;  
import java.util.List;  
  
import com.company.model.Student;  
import com.mysql.jdbc.exceptions.MySQLIntegrityConstraintViolationException;  
import java.math.BigDecimal;  
import java.sql.Date;  
  
public class StudentDao {  
  
 Connection con = null;  
 PreparedStatement ps = null;  
 ResultSet rs = null;  
 int st;//status  
  
 public int insert(Student student) {  
 con = ConnectionFactory.*getConnection*();  
 try {  
 String query = "insert into student(fname,lname,address,mobile\_no,email\_id,city,designation,dob,doj,salary) "  
 + "values(?,?,?,?,?,?,?,?,?,?)";  
 ps = con.prepareStatement(query);  
 ps.setString(1, student.getFname());  
 ps.setString(2, student.getLname());  
 ps.setString(3, student.getAddress());  
 ps.setString(4, student.getMobileNo());  
 ps.setString(5, student.getMailId());  
 ps.setString(6, student.getCity());  
 ps.setString(7, student.getDesignation());  
 ps.setDate(8, student.getDob());  
 ps.setDate(9, student.getDoj());  
 ps.setBigDecimal(10, student.getSalary());  
 st = ps.executeUpdate();  
 System.*out*.println("inserted student " + st);  
 } catch (MySQLIntegrityConstraintViolationException e) {  
 st = -1;  
 e.printStackTrace();  
 } catch (Exception e) {  
 st = -2;  
 e.printStackTrace();  
 } finally {  
 try {  
 con.close();  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 }  
 }  
 return st;  
 }  
  
 public int update(Student student) {  
 con = ConnectionFactory.*getConnection*();  
 try {  
 String query = "update student set fname=?,lname=?,address=?,mobile\_no=?,email\_id=?,city=?, "  
 + "designation=?,dob=?,doj=?,salary=? "  
 + "where id=? ";  
 ps = con.prepareStatement(query);  
 ps.setString(1, student.getFname());  
 ps.setString(2, student.getLname());  
 ps.setString(3, student.getAddress());  
 ps.setString(4, student.getMobileNo());  
 ps.setString(5, student.getMailId());  
 ps.setString(6, student.getCity());  
 ps.setString(7, student.getDesignation());  
 ps.setDate(8, student.getDob());  
 ps.setDate(9, student.getDoj());  
 ps.setBigDecimal(10, student.getSalary());  
 ps.setLong(11, student.getId());  
 st = ps.executeUpdate();  
 System.*out*.println("updated student " + st);  
 } catch (Exception e) {  
 st = -2;  
 e.printStackTrace();  
 } finally {  
 try {  
 con.close();  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 }  
 }  
 return st;  
 }  
  
 public int delete(long id) {  
 con = ConnectionFactory.*getConnection*();  
 try {  
 String query = "delete from student where id=? ";  
 ps = con.prepareStatement(query);  
 ps.setLong(1, id);  
 st = ps.executeUpdate();  
 System.*out*.println("deleted student " + st);  
 } catch (Exception e) {  
 st = -2;  
 e.printStackTrace();  
 } finally {  
 try {  
 con.close();  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 }  
 }  
 return st;  
 }  
  
 public Student fetchById(long id) {  
 Student student = new Student();  
 con = ConnectionFactory.*getConnection*();  
 try {  
 String query = "select \* from student where id=?";  
 ps = con.prepareStatement(query);  
 ps.setLong(1, id);  
 rs = ps.executeQuery();  
 while (rs.next()) {  
 student.setId(rs.getLong("id"));  
 student.setFname(rs.getString("fname"));  
 student.setLname(rs.getString("lname"));  
 student.setAddress(rs.getString("address"));  
 student.setMobileNo(rs.getString("mobile\_no"));  
 student.setMailId(rs.getString("email\_id"));  
 student.setCity(rs.getString("city"));  
 student.setDesignation(rs.getString("designation"));  
 student.setDob(rs.getDate("dob"));  
 student.setDoj(rs.getDate("doj"));  
 student.setSalary(rs.getBigDecimal("salary"));  
 student.setAddDate(rs.getTimestamp("add\_date"));  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 } finally {  
 try {  
 con.close();  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 }  
 }  
 return student;  
 }  
  
 public Student fetchByEmailId(String emailId) {  
 Student student = new Student();  
 con = ConnectionFactory.*getConnection*();  
 try {  
  
  
 // Найти студентов по emailID  
  
  
  
 } catch (Exception e) {  
 e.printStackTrace();  
 } finally {  
 try {  
 con.close();  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 }  
 }  
 return student;  
 }  
  
 public Student fetchByMobileNo(String mobileNo) {  
 Student student = new Student();  
 con = ConnectionFactory.*getConnection*();  
 try {  
  
  
  
 // Найти студентов по Номеру телефона  
  
  
  
 } catch (Exception e) {  
 e.printStackTrace();  
 } finally {  
 try {  
 con.close();  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 }  
 }  
 return student;  
 }  
  
 public List<Student> searchByName(String name) {  
 List<Student> studentList = new ArrayList<Student>();  
  
 con = ConnectionFactory.*getConnection*();  
 try {  
  
 // Найти студентов по Имени  
 // Модернизируйте метод. С возможностью искать и по имени и по фамилии с помощью слова like  
  
  
 } catch (Exception e) {  
 e.printStackTrace();  
 } finally {  
 try {  
 con.close();  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 }  
 }  
 return studentList;  
 }  
  
 public List<Student> fetchByCity(String city) {  
 List<Student> studentList = new ArrayList<Student>();  
  
 con = ConnectionFactory.*getConnection*();  
 try {  
 String query = "select \* from student where city=?";  
 ps = con.prepareStatement(query);  
 ps.setString(1, city);  
 rs = ps.executeQuery();  
 while (rs.next()) {  
 Student student = new Student();  
 student.setId(rs.getLong("id"));  
 student.setFname(rs.getString("fname"));  
 student.setLname(rs.getString("lname"));  
 student.setAddress(rs.getString("address"));  
 student.setMobileNo(rs.getString("mobile\_no"));  
 student.setMailId(rs.getString("email\_id"));  
 student.setCity(rs.getString("city"));  
 student.setDesignation(rs.getString("designation"));  
 student.setDob(rs.getDate("dob"));  
 student.setDoj(rs.getDate("doj"));  
 student.setSalary(rs.getBigDecimal("salary"));  
 student.setAddDate(rs.getTimestamp("add\_date"));  
 studentList.add(student);  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 } finally {  
 try {  
 con.close();  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 }  
 }  
 return studentList;  
 }  
  
 public List<Student> fetchBySalaryRange(BigDecimal lowerSalary, BigDecimal higherSalary) {  
 List<Student> studentList = new ArrayList<Student>();  
  
 con = ConnectionFactory.*getConnection*();  
 try {  
  
  
  
 // Найти студентов с ЗП в пределах От и До  
  
  
  
 } catch (Exception e) {  
 e.printStackTrace();  
 } finally {  
 try {  
 con.close();  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 }  
 }  
 return studentList;  
 }  
  
 public List<Student> fetchByDob(Date dob) {  
 List<Student> studentList = new ArrayList<Student>();  
  
 con = ConnectionFactory.*getConnection*();  
 try {  
 String query = "select \* from student where dob=?";  
 ps = con.prepareStatement(query);  
 ps.setDate(1, dob);  
 rs = ps.executeQuery();  
 while (rs.next()) {  
 Student student = new Student();  
 student.setId(rs.getLong("id"));  
 student.setFname(rs.getString("fname"));  
 student.setLname(rs.getString("lname"));  
 student.setAddress(rs.getString("address"));  
 student.setMobileNo(rs.getString("mobile\_no"));  
 student.setMailId(rs.getString("email\_id"));  
 student.setCity(rs.getString("city"));  
 student.setDesignation(rs.getString("designation"));  
 student.setDob(rs.getDate("dob"));  
 student.setDoj(rs.getDate("doj"));  
 student.setSalary(rs.getBigDecimal("salary"));  
 student.setAddDate(rs.getTimestamp("add\_date"));  
 studentList.add(student);  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 } finally {  
 try {  
 con.close();  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 }  
 }  
 return studentList;  
 }  
  
 public List<Student> fetchByRangeDoj(Date startDate, Date endDate) {  
 List<Student> studentList = new ArrayList<Student>();  
  
 con = ConnectionFactory.*getConnection*();  
 try {  
  
  
 // Найти студентов с датами От и До  
  
  
  
  
 } catch (Exception e) {  
 e.printStackTrace();  
 } finally {  
 try {  
 con.close();  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 }  
 }  
 return studentList;  
 }  
  
 public List<Student> fetchAll() {  
 List<Student> studentList = new ArrayList<Student>();  
  
 con = ConnectionFactory.*getConnection*();  
 try {  
  
 // Модернизируйте поиск - отсортировав от большего к меньшему по полю id  
  
  
 String query = "select \* from student";  
 ps = con.prepareStatement(query);  
 rs = ps.executeQuery();  
 while (rs.next()) {  
 Student student = new Student();  
 student.setId(rs.getLong("id"));  
 student.setFname(rs.getString("fname"));  
 student.setLname(rs.getString("lname"));  
 student.setAddress(rs.getString("address"));  
 student.setMobileNo(rs.getString("mobile\_no"));  
 student.setMailId(rs.getString("email\_id"));  
 student.setCity(rs.getString("city"));  
 student.setDesignation(rs.getString("designation"));  
 student.setDob(rs.getDate("dob"));  
 student.setDoj(rs.getDate("doj"));  
 student.setSalary(rs.getBigDecimal("salary"));  
 student.setAddDate(rs.getTimestamp("add\_date"));  
 studentList.add(student);  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 } finally {  
 try {  
 con.close();  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 }  
 }  
 return studentList;  
 }  
}

Класс **ConnectionFactory**

package com.company.dao;  
  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.SQLException;  
  
public class ConnectionFactory {  
  
 public static Connection getConnection() {  
 Connection c = null;  
 try {  
 Class.*forName*("com.mysql.jdbc.Driver");  
 c = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/ibapst?useUnicode=true&characterEncoding=UTF-8", "root", "root");  
 } catch (ClassNotFoundException e) {  
 System.*out*.println("ClassNotFoundException " + e);  
 } catch (SQLException e) {  
 System.*out*.println("SQLException " + e);  
 }  
 return c;  
 }  
}

Класс **StudentAction**

package com.company.action;  
  
import com.company.dao.StudentDao;  
import com.company.model.Student;  
  
import java.math.BigDecimal;  
import java.sql.Date;  
import java.util.List;  
  
  
public class StudentAction {  
  
 StudentDao dao = new StudentDao();  
 int st;  
  
 public void insert(Student student) {  
 st = dao.insert(student);  
 if (st == 1) {  
 System.*out*.println("Student Inserted Successfully");  
 } else if (st == -1) {  
 System.*out*.println("Student Already exists");  
 } else {  
 System.*out*.println("Unable to Insert Student");  
 }  
 }  
  
 public void update(Student student) {  
 st = dao.update(student);  
 if (st == 1) {  
 System.*out*.println("Student Updated Successfully");  
 } else {  
 System.*out*.println("Unable to update Student");  
 }  
 }  
  
 public void delete(Long id) {  
 st = dao.delete(id);  
 if (st == 1) {  
 System.*out*.println("Student Deleted Successfully");  
 } else {  
 System.*out*.println("No Record Found");  
 }  
 }  
  
 public void fetchById(Long id) {  
 Student student = dao.fetchById(id);  
 if (student.getId() == 0) {  
 System.*out*.println("No Record Found");  
 } else {  
 System.*out*.println("Student Details are :");  
 System.*out*.println(student);  
 }  
 }  
  
 public void fetchByEmailId(String emailId) {  
 Student student = dao.fetchByEmailId(emailId);  
 if (student.getId() == 0) {  
 System.*out*.println("No Record Found");  
 } else {  
 System.*out*.println("Student Details are :");  
 System.*out*.println(student);  
 }  
 }  
  
 public void fetchByMobileNo(String mobileNo) {  
 Student student = dao.fetchByMobileNo(mobileNo);  
 if (student.getId() == 0) {  
 System.*out*.println("No Record Found");  
 } else {  
 System.*out*.println("Student Details are :");  
 System.*out*.println(student);  
 }  
 }  
  
 public void searchByName(String name) {  
 List<Student> studentList = dao.searchByName(name);  
 if (studentList.isEmpty()) {  
 System.*out*.println("No Record Found");  
 } else {  
 System.*out*.println("Student Details are :");  
 for (Student student : studentList) {  
 System.*out*.println(student);  
 }  
 }  
  
 }  
  
 public void fetchByCity(String city) {  
 List<Student> studentList = dao.fetchByCity(city);  
 if (studentList.isEmpty()) {  
 System.*out*.println("No Record Found");  
 } else {  
 System.*out*.println("Student Details are :");  
 for (Student student : studentList) {  
 System.*out*.println(student);  
 }  
 }  
 }  
  
 public void fetchBySalaryRange(BigDecimal lowerSalary, BigDecimal higherSalary) {  
 List<Student> studentList = dao.fetchBySalaryRange(lowerSalary, higherSalary);  
 if (studentList.isEmpty()) {  
 System.*out*.println("No Record Found");  
 } else {  
 System.*out*.println("Student Details are :");  
 for (Student student : studentList) {  
 System.*out*.println(student);  
 }  
 }  
 }  
  
 public void fetchByDob(Date dob) {  
 List<Student> studentList = dao.fetchByDob(dob);  
 if (studentList.isEmpty()) {  
 System.*out*.println("No Record Found");  
 } else {  
 System.*out*.println("Student Details are :");  
 for (Student student : studentList) {  
 System.*out*.println(student);  
 }  
 }  
 }  
  
 public void fetchByRangeDoj(Date startDate, Date endDate) {  
 List<Student> studentList = dao.fetchByRangeDoj(startDate, endDate);  
 if (studentList.isEmpty()) {  
 System.*out*.println("No Record Found");  
 } else {  
 System.*out*.println("Student Details are :");  
 for (Student student : studentList) {  
 System.*out*.println(student);  
 }  
 }  
 }  
  
 public void fetchAll() {  
 List<Student> studentList = dao.fetchAll();  
 if (studentList.isEmpty()) {  
 System.*out*.println("No Record Found");  
 } else {  
 System.*out*.println("Student Details are :");  
 for (Student student : studentList) {  
 System.*out*.println(student);  
 }  
 }  
 }  
  
}

Задача.

Начиная с класса main пройти по цепочке вызова методов и восстановить недостающие фрагменты кода.

8

1. В классе MainClass определить возов методов и проверить их реализацию. Определить вызываемый метод в StudentAction
2. В классе StudentAction проверить реализацию соответствующего метода.
3. В классе StudentDao определить недостающие методы для соответствующих действий пользователя.

9-10

1. Добавить еще 1 сущность и выполнить несколько операций (2-4).