JDBC
DERBY
DAO

JDBC (Java DataBase Connectivity) - набор библиотек для работы с базами данных.

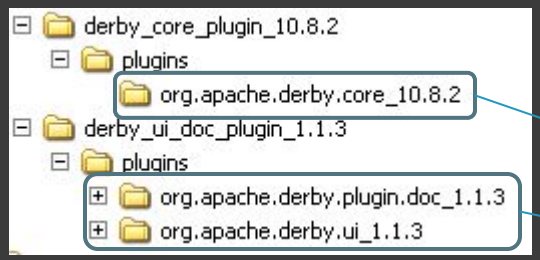
Apache Derby - реляционная СУБД, написана на Java.
Страница с релизами: http://db.apache.org/derby/releases/

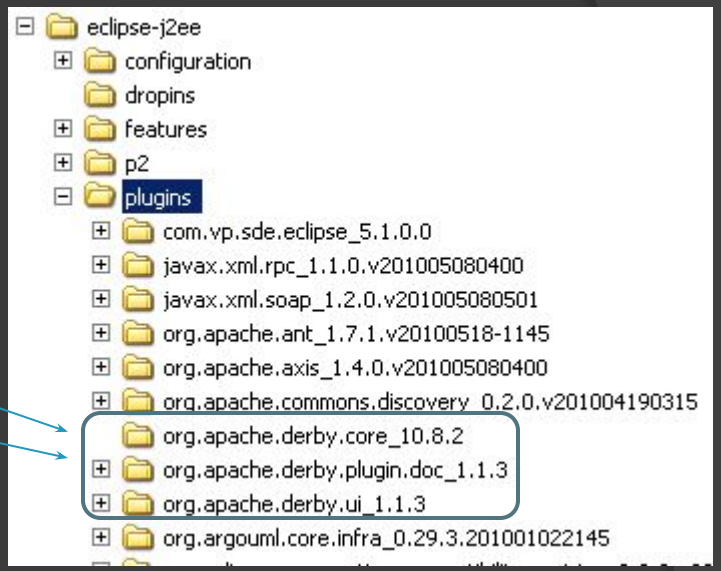
Eclipse позволяет запускать и работать с Derby с помощью плагинов: derby_core_plugin и derby_ui_doc_plugin

Адреса для загрузки плагинов (последний релиз):
http://archive.apache.org/dist/db/derby/db-derby-10.8.2.2/derby_core_plugin_10.8.2.zip
http://archive.apache.org/dist/db/derby/db-derby-10.8.2.2/derby_ui_doc_plugin_1.1.3.zip

Скачиваем. Распаковываем.

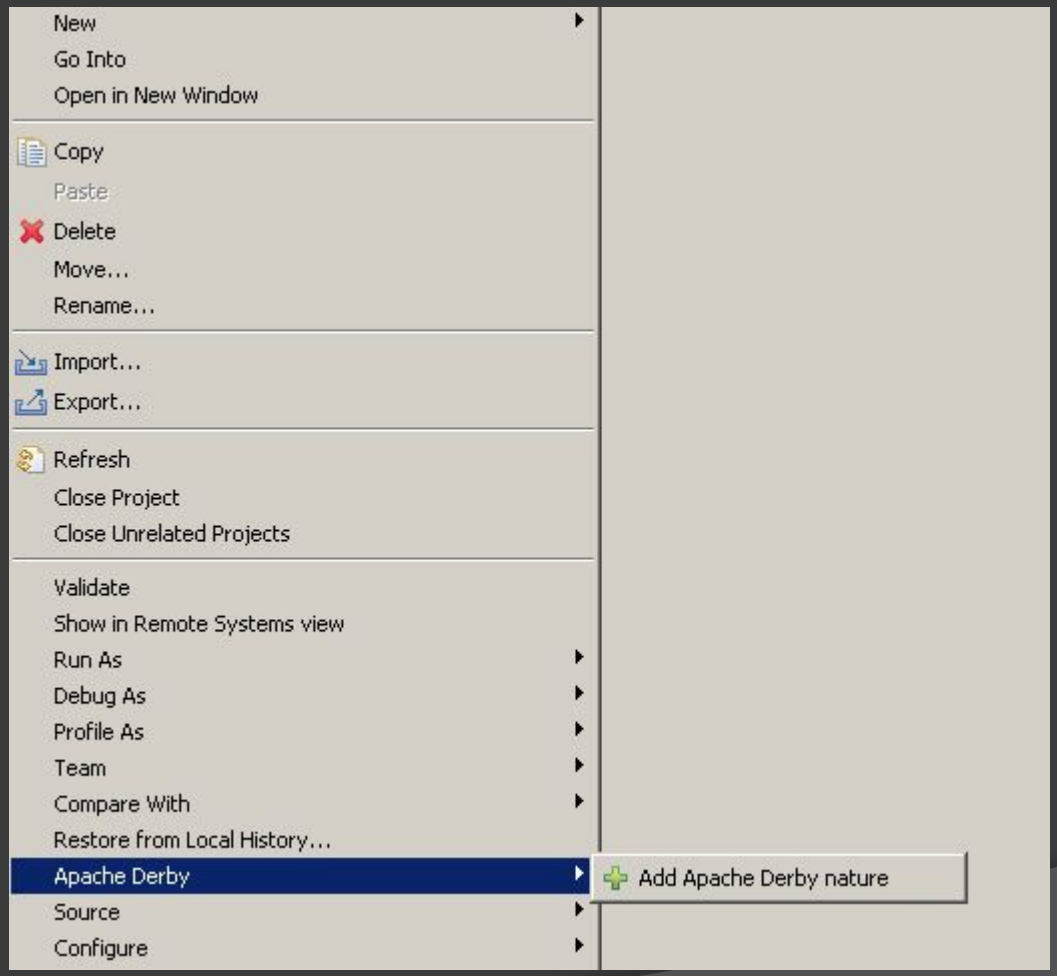
Для установки плагинов нужно скопировать содержимое каталогов plugins в каталог plugins инсталяции Eclipse



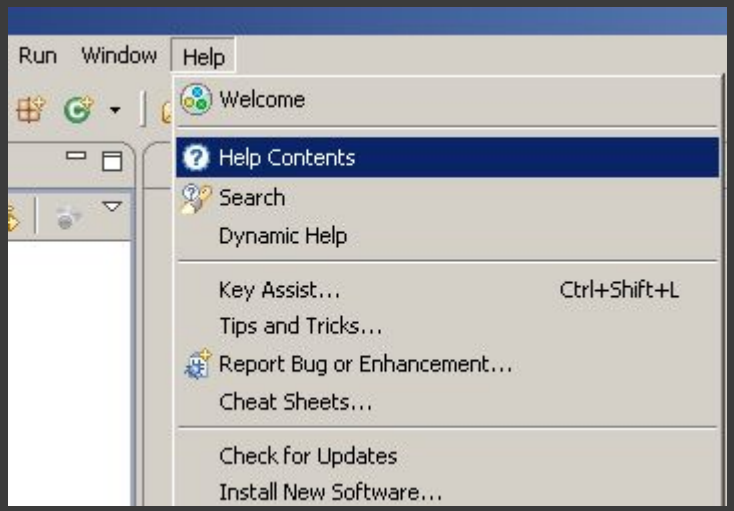


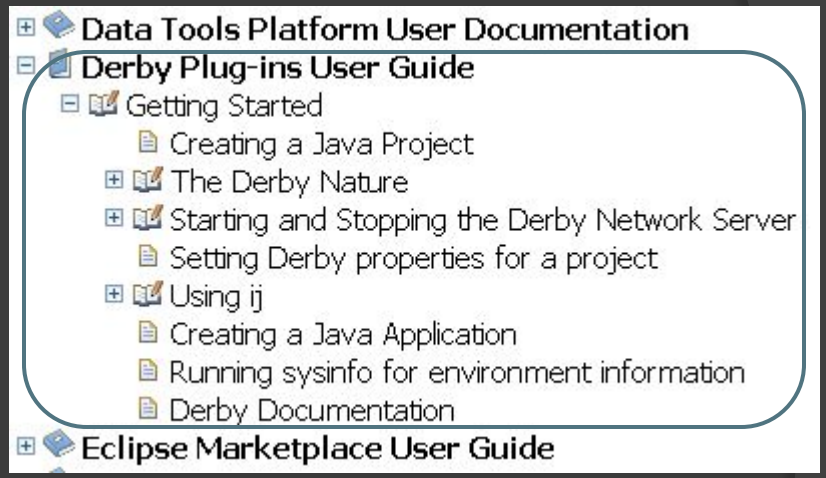
Перегружаем Eclipse.

После перегрузки, плагин установлен и его функциональность доступна через пункт контекстного меню.



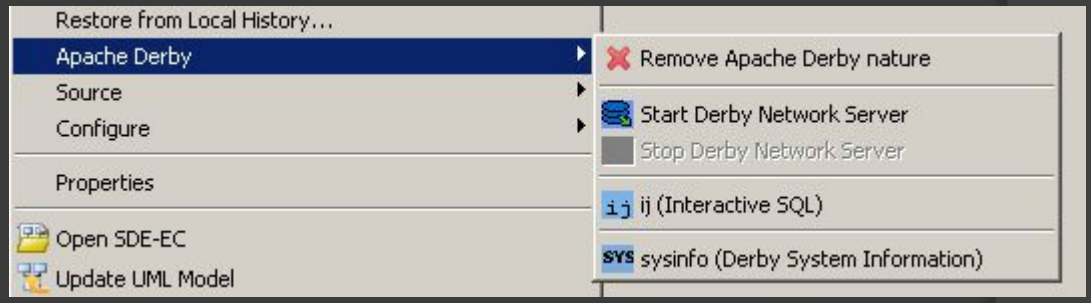
Применение плагина документировано.



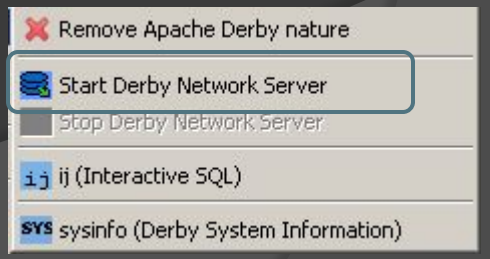


Для того чтобы добавить возможность работать с Derby из вашего проекта нужно на проекте нажать ПКМ (правая клавиша мыши) и выбрать пункт меню
Add Apache Derby nature.

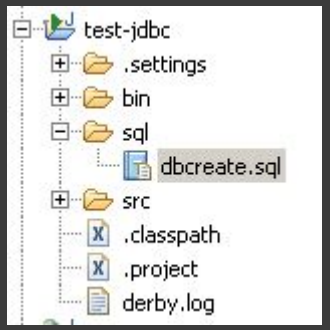
После этого будут доступны новые пункты в меню



Для запуска СУБД нужно выбрать
Start Derby Network Server



Создаем в проекте каталог sql и в нем файл dbcreate.sql



В базе будет две таблицы: roles (роли)
В ней три записи, как указанно

ID	NAME
1	admin
2	tutor
3	student

и users (пользователи)

ID	LOGIN	PASSWORD HASH	FIRST NAME	LAST NAME	ROLE ID
1	...	...	...	...	...

в таблице users записи заранее не предопределены (см. далее).

Для того, чтобы создать пустую базу данных необходимо выполнить следующую команду:

```
connect 'jdbc:derby://localhost:1527/ts;create=true;user=ts;password=ts';
```

connect
команда соединения с СУБД Derby

'jdbc:derby://localhost:1527/ts;create=true;user=ts;password=ts'
строка соединения

jdbc:derby://
префикс

localhost:1527
адрес, где запущена СУБД и порт, на котором она принимает соединения

connect 'jdbc:derby://localhost:1527/ts;create=true;user=ts;password=ts';

ts

имя базы данных

create=true;user=ts;password=ts

параметры соединения:

create=true

при соединении создать новую базу данных, если она отсутствует

user=ts;password=ts

логин и пароль пользователя, который имеет право работать в последствие с указанной базой данных

В файле dbcreate.sql записываем:

```
connect 'jdbc:derby://localhost:1527/ts;create=true;user=ts;password=ts';
```

сохраняем файл. Далее ПКМ на файле и выбираем

(сама СУБД должна быть, разумеется, запущена)
В результате выполнения скрипта будет создана пустая база данных ts. Вывод в консоль при выполнении скрипта:

```
ij version 10.8
ij> connect 'jdbc:derby://localhost:1527/ts;create=true;user=ts;password=ts';
ij>
```

Для того, чтобы создать таблицу roles в базе данных ts достаточно модифицировать исходный скрипт:

```
connect 'jdbc:derby://localhost:1527/ts;user=ts;password=ts';

CREATE TABLE roles(
    id INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    name VARCHAR(20) NOT NULL UNIQUE
);
```

Параметр create=true можно не удалять из строки соединения. Если база данных существует, то этот параметр будет проигнорирован.

Таблица roles состоит из двух полей: id и name.
Поле id объявлено так:

```
id INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
```

INTEGER (или INT) – это тип данных (целое число);

NOT NULL – означает, что для любой строки в таблице поле id не должно быть пустым, т.е. туда должно быть что то записано;

GENERATED ALWAYS AS IDENTITY - означает, что поле будет автоматически инкрементировано самой СУБД при добавлении новой строки, начиная с нуля и с шагом 1; при этом "вручную" вставить значение в это поле невозможно (подробно тут: http://db.apache.org/derby/docs/dev/ref/rrefsqlj37836.html);

PRIMARY KEY - означает, что данное поле является первичным ключом, по значению поля id может быть однозначно идентифицирована любая строка в таблице.

Поле name объявлено так: 

VARCHAR(20) – строковый тип данных (не более 20 символов);

NOT NULL – значение поля не должно быть пустым;

UNIQUE – значение поля должно быть уникальным, т.е. не допускается существование двух разных строк с одинаковым значением этого поля.

Модифицируем исходный скрипт:

```
connect 'jdbc:derby://localhost:1527/ts;create=true;user=ts;password=ts';

DROP TABLE roles;

CREATE TABLE roles(
    id INT NOT NULL GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    name VARCHAR(20) NOT NULL UNIQUE
);

INSERT INTO roles (name) VALUES ('admin');
INSERT INTO roles (name) VALUES ('tutor');
INSERT INTO roles (name) VALUES ('student');
```

Команда DROP TABLE удаляет указанную таблицу (нужна, чтобы не было ошибки создания уже существующей таблицы на следующем шаге);

Команда INSERT вставляет данные в таблицу.

Чтобы проверить правильность внесения данных в таблицу roles, достаточно выполнить скрипт (select.sql):

```
connect 'jdbc:derby://localhost:1527/ts;user=ts;password=ts';

SELECT * FROM roles;
```

Результат выполнения:

```
ij version 10.8
ij> connect 'jdbc:derby://localhost:1527/ts;user=ts;password=ts';
ij> SELECT * FROM roles;
ID          |NAME
------------------------------
1           |admin
2           |tutor
3           |student

3 rows selected
ij>
```

СУБД Derby запущена, в базе данных ts одна таблица roles. Для того, чтобы иметь возможность обмениваться информацией с БД, можно использовать JDBC.

Вначале создадим java-bean класс Role (в отдельном пакете entity), который будет соответствовать одной записи в таблице roles:

```
package ua.kharkov.knure.testDerby.db.entity;

public class Role {
    private int id;
    private String name;

    public int getId() {return id;}
    public void setId(int id) {this.id = id;}

    public String getName() {return name;}
    public void setName(String name) {this.name = name;}

    public String toString() {
        return new StringBuffer("Role [id=").append(getId()).append(", ")
        .append("name=").append(name).append("]")
        .toString();
    }
}
```

Создадим абстрактный класс DAOFactory

```
package ua.kharkov.knure.testDerby.db.dao;

import java.sql.*;

import ua.kharkov.knure.testDerby.db.Constants;

public abstract class DAOFactory {
    private static DAOFactory instance = null;

    public static synchronized DAOFactory getInstance() {
        if (instance == null) {
            try {
                Class.forName(Constants.DRIVER_CLASS_NAME);
                Class clazz = Class.forName(Constants.DAO_FACTORY_CLASS_NAME);
                instance = (DAOFactory) clazz.newInstance();
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }
        return instance;
    }

    public abstract RoleDAO getRoleDAO();

    public static Connection getConnection() throws SQLException {
            Connection connection = null;
            connection = DriverManager.getConnection(Constants.DB_URL,
                    Constants.LOGIN, Constants.PASSWORD);
            return connection;
    }
}
```

Класс DAOFactory предназначен для получения соединения с базой данных, с какой именно базой данных будет соединение будет определено значениями констант из класса Constants:

```
package ua.kharkov.knure.testDerby.db;

public final class Constants {
    public static final String DRIVER_CLASS_NAME = "org.apache.derby.jdbc.ClientDriver";
    public static final String DB_URL = "jdbc:derby://localhost:1527/ts";
    public static final String DAO_FACTORY_CLASS_NAME =
        "ua.kharkov.knure.testDerby.db.dao.derby.DerbyDAOFactory";
    public static final String LOGIN = "ts";
    public static final String PASSWORD = "ts";
}
```

В реальном проекте эти значения будут взяты из внешнего конфигурационного файла.

Класс DAOFactory:

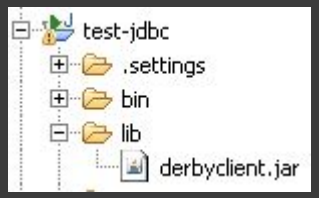
```
Class.forName(Constants.DRIVER_CLASS_NAME);
```

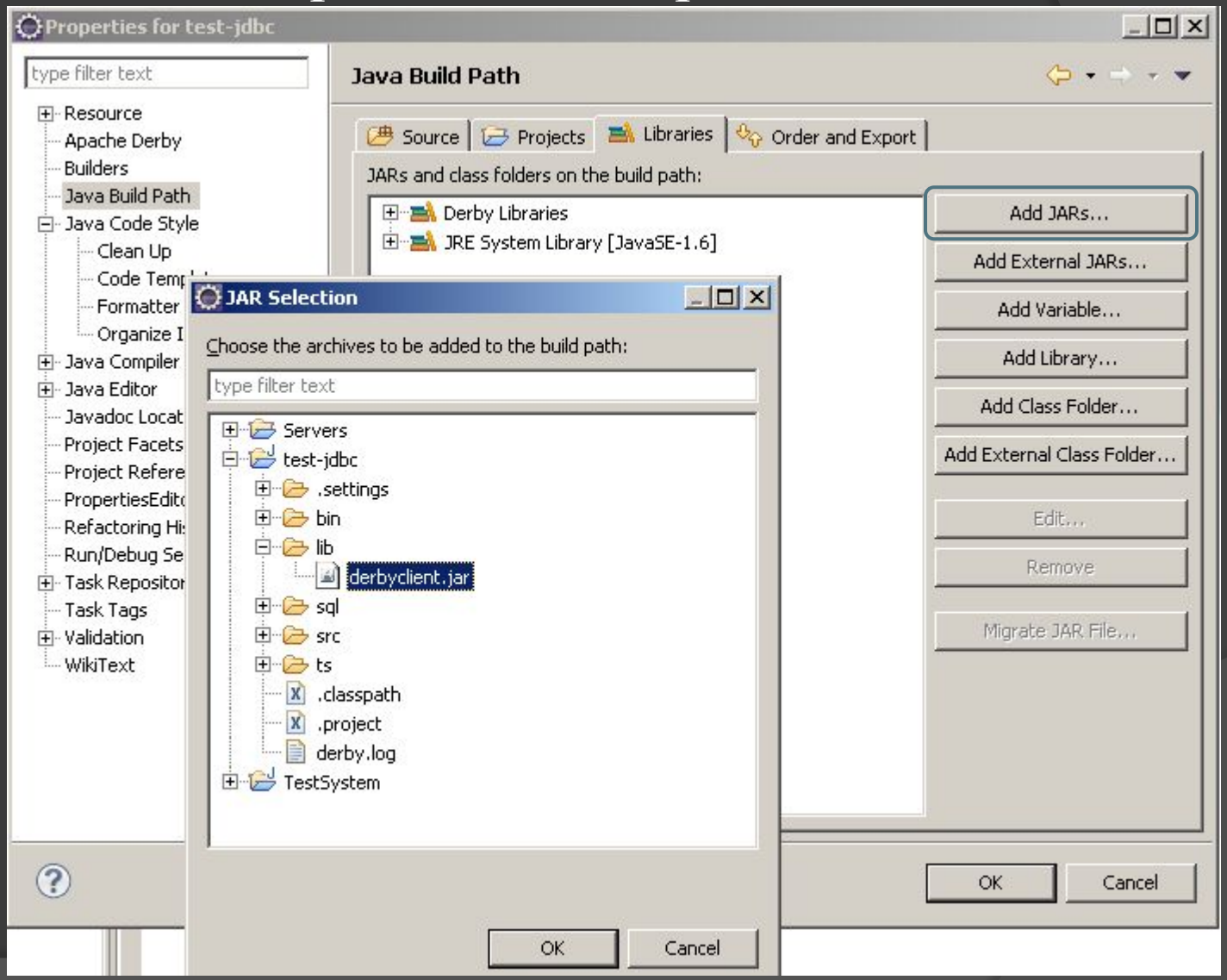
Данная строка регистрирует драйвер СУБД в JVM. В нашем случае это драйвер

org.apache.derby.jdbc.ClientDriver

Для того, чтобы приложение могло работать с ним, нужно к проекту подключить библиотеку derbyclient.jar.

Файл derbyclient.jar находится в поставке СУБД Derby. Можно создать каталог lib внутри проекта, скопировать туда этот файл и подключить через свойства проекта:





Класс DAOFactory:

```
Class clazz = Class.forName(Constants.DAO_FACTORY_CLASS_NAME);
instance = (DAOFactory) clazz.newInstance();
```

Данные строки создают экземпляр класса по его FCN – (full qualified name), т.е. полному имени.

Метод getConnection

```
public static Connection getConnection() throws SQLException {
        Connection connection = null;
        connection = DriverManager.getConnection(Constants.DB_URL,
                Constants.LOGIN, Constants.PASSWORD);
        return connection;
}
```

Возвращает объект Connection – соединение с базой данных.

Класс DAOFactory работает с интерфейсами, которые не зависят от конкретной СУБД.

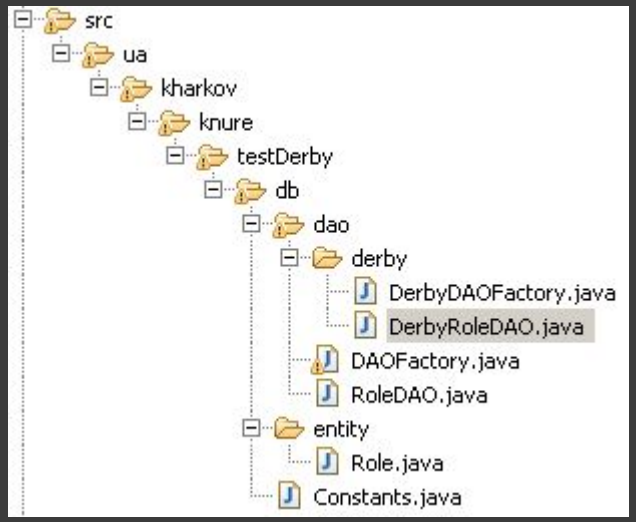
```
package ua.kharkov.knure.testDerby.db.dao;

import java.util.List;
import ua.kharkov.knure.testDerby.db.entity.Role;

public interface RoleDAO {
    Role findRoleById(Integer id);
    List<Role> findAllRoles();
}
```

Методы

```
Role findRoleById(Integer id);
List<Role> findAllRoles();
```

представляют собой логику работы с базой данных.

Реализация DAO для СУБД Derby находится в пакете ua.kharkov.knure.testDerby.db.dao.derby



Каждой поддерживаемой СУБД будет соответствовать набор классов в пакете ua.kharkov.knure.testDerby.db.dao.ИМЯ_БАЗЫ_ДАННЫХ

Класс DerbyDAOFactory, представляет собой реализацию DAOFactory для СУБД Derby:

```
package ua.kharkov.knure.testDerby.db.dao.derby;

import ua.kharkov.knure.testDerby.db.dao.*;

public class DerbyDAOFactory extends DAOFactory {
    public RoleDAO getRoleDAO() {
        return new DerbyRoleDAO();
    }
}
```

Реализация логики находится в классе DerbyRoleDAO:

```
package ua.kharkov.knure.testDerby.db.dao.derby;

import java.sql.*;
import java.util.*;

import ua.kharkov.knure.testDerby.db.dao.*;
import ua.kharkov.knure.testDerby.db.entity.*;


public class DerbyRoleDAO implements RoleDAO {
    private static final String SQL__SELECT_FROM_ROLE_BY_ID = "SELECT * FROM roles WHERE id = ?";
    private static final String SQL__SELECT_FROM_ROLE = "SELECT * FROM roles";
```

Метод findRoleById:

```
public Role findRoleById(Integer id) {
    Connection connection = null;
    Role role = null;
    try {
        connection = DAOFactory.getConnection();
        PreparedStatement pstmt = connection.prepareStatement(SQL__SELECT_FROM_ROLE_BY_ID);
        pstmt.setInt(1, id);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) role = extractRole(rs);
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
    return role;
}
```

Метод findAllRoles:

```
public List<Role> findAllRoles() {
    List<Role> roleList = new ArrayList<Role>();
    Connection connection = null;
    Statement stmt = null;
    ResultSet resultSet = null;
    try {
        connection = DAOFactory.getConnection();
        stmt = connection.createStatement();
        resultSet = stmt.executeQuery(SQL__SELECT_FROM_ROLE);
        while (resultSet.next()) {
            roleList.add(extractRole(resultSet));
        }
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
    return roleList;
}
```

Метод утилита:

```
    private Role extractRole(ResultSet rs) throws SQLException {
        Role role = new Role();
        role.setId(rs.getInt("id"));
        role.setName(rs.getString("name"));
        return role;
    }
}
```

Загружает по строке таблицы объект Role.