DATABASE ASSIGNMENT

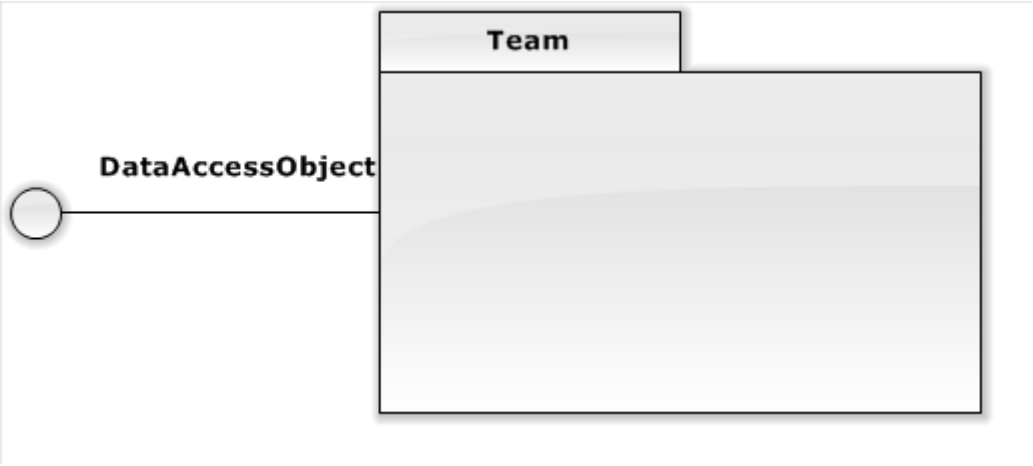
BRIEF

This is a study point assignment (5 study points). You work in pairs.

You must write a Java program that connects to a database vía JDBC and retrieves data and displays it in the console. Appendix A is an overview of the database design. Appendix B is the data.

The program you write will be used as the backend part of a larger system, so you don't have to worry about designing a user interface. All you will have to do is to implement a set of methods defined in a Java interface which will act as the contract between the user interface and the backend part of the system.

Figure 1 is a high level view of the system you are about to build. The UML package represents the system, but abstracts away the classes (which will be located inside the package). The lollipop (ball on a stick) illustrates an interface called DataAccessObject that the component provides.



FIGUR 1 UML PACKAGE DIAGRAM OF THE BACKEND COMPONENT YOU MUST IMPLEMENT

CONTENTS

Brief....1
The database....3
ResultSet....3
Testing....4
Hand in....4
What to hand in....4
When to hand in....4
Where to hand in....4
Appendix A – ER diagram....5
Appendix B - Data....6
Appendix C – Script....7
Appendix D - DBConnector....8
Appendix E – Data Access Interface....8
Appendix F – JUnit....9

THE DATABASE

In the appendix (Appendix C) you will find a script which will create a database. The data you need to insert can be found in Appendix B.

(The script only creates the tables – First you must create a new database schema into which the tables will be created).

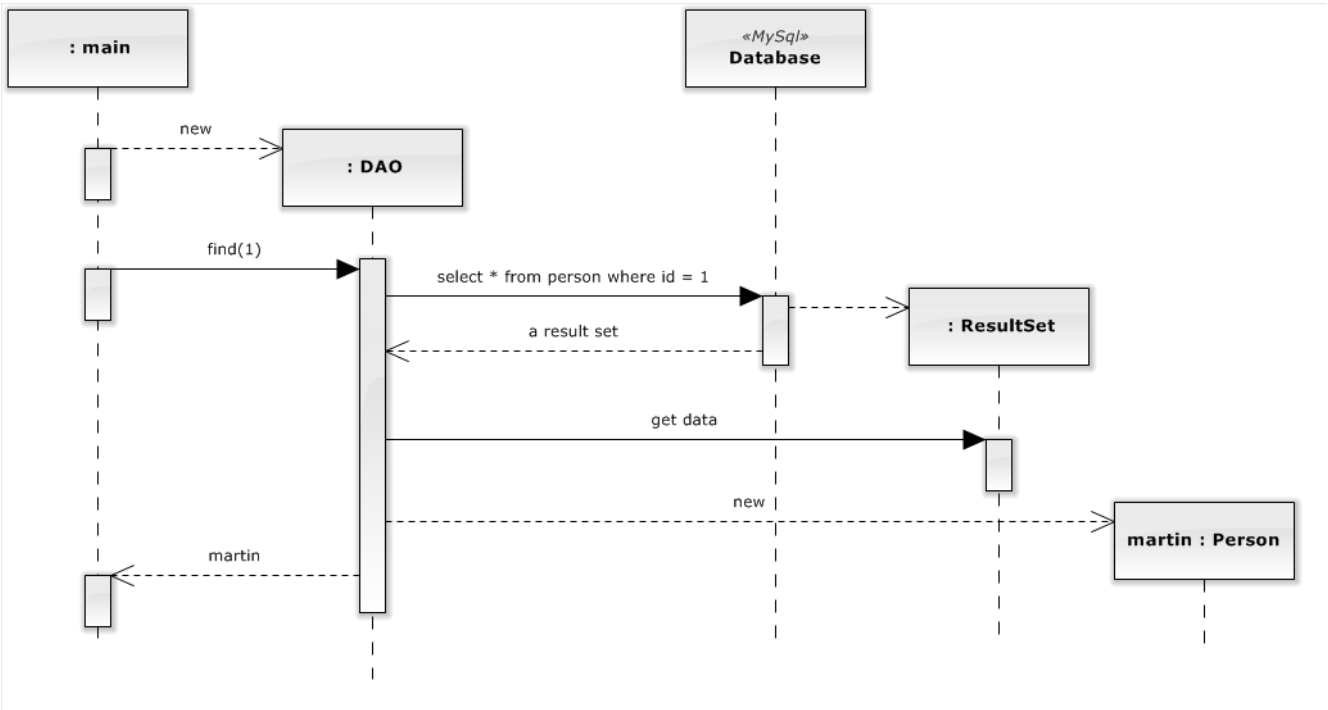
RESULTSET

You can use the Connector in Appendix D or use your own database connector if you already have one.

You will have to write entity classes for both "User", "Team" and a "Data Access Object" class.

The "Data Access Object" class interprets ResultSets and returns entities.

The entities must have toString() methods and the "Data Access Object" class must implement the interface in Appendix E.



THIS DIAGRAM SHOWS THE PRINCIPLES OF WHAT YOU ARE ABOUT TO DO, BUT NOT EXACTLY WHAT YOU ARE ABOUT TO DO.

(There is an argument for making more than one "Data Access Object" class, but in this case we will settle for just one.)

TESTING

Appendix F contains a JUnit which tests some of the methods in the “Data Access Object”. You must make these tests pass. Also you need to write additional test cases, so that 100% of the methods in “Data Access Object” are covered by tests.

HAND IN

WHAT TO HAND IN

Export the finished Netbeans project to a .zip file.

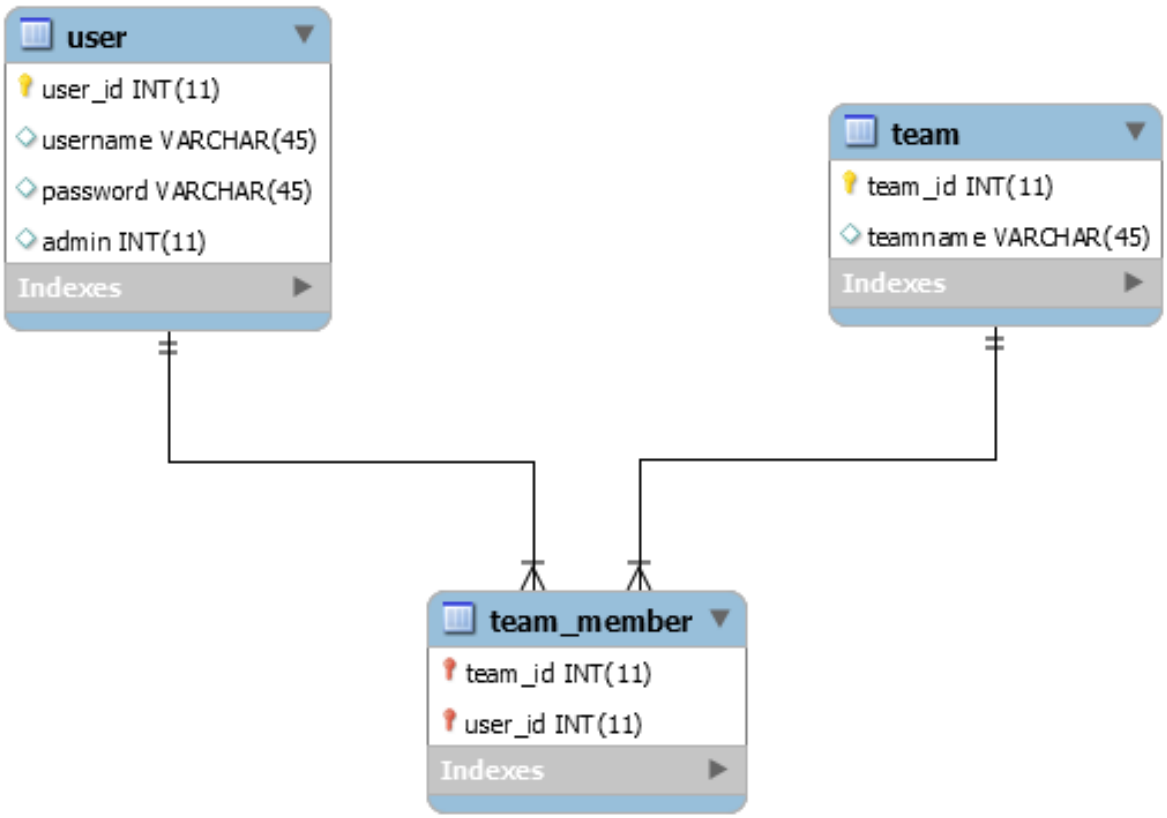
WHEN TO HAND IN

Hand in on Friday November 18th 2016 before 23:59

WHERE TO HAND IN

Hand in on Fronter as a group hand-in (2 students per group)

APPENDIX A – ER DIAGRAM



FIGUR 2

APPENDIX B - DATA

This is the data as the customer sees it. It is not database tables. Tables should never have more than one value in a cell. So you must redesign according to ER diagram in appendix A.

User			
ID	USERNAME	PASSWORD	ADMIN
1	Anders And	1234	true
2	Mickey Mouse	5678	true
3	Fedtmule	1234	false
4	George Gearløs	1234	false
5	Bedstemor And	1234	false
6	Onkel Joakim	1234	false
7	Pluto	1234	false
8	Fætter Guf	1234	false

FIGURE 1 USER DATA

Team		
ID	TEAMNAME	TEAM MEMBERS
1	A	- Mickey Mouse - Fedtmule - Pluto
2	B	- Anders And - George Gearløs - Bedstemor And - Onkel Joakim - Fætter Guf
3	C	- Anders And - Mickey Mouse - Fedtmule - Pluto

FIGURE 3 TEAM DATA

APPENDIX C – SCRIPT

```
DROP TABLE IF EXISTS `team_member`;
DROP TABLE IF EXISTS `team`;
DROP TABLE IF EXISTS `user`;

CREATE TABLE `team` (
 `team_id` int(11) NOT NULL,
 `teamname` varchar(45),
 PRIMARY KEY (`team_id`)
);

CREATE TABLE `user` (
 `user_id` int(11) NOT NULL,
 `username` varchar(45),
 `password` varchar(45),
 `admin` int(11) DEFAULT NULL,
 PRIMARY KEY (`user_id`)
);

CREATE TABLE `team_member` (
 `team_id` int(11) NOT NULL,
 `user_id` int(11) NOT NULL,
 PRIMARY KEY (`team_id`,`user_id`),
 CONSTRAINT `fk_team` FOREIGN KEY (`team_id`) REFERENCES `team` (`team_id`),
 CONSTRAINT `fk_user` FOREIGN KEY (`user_id`) REFERENCES `user` (`user_id`)
);
```

APPENDIX D - DBCONNECTOR

```
public class DBConnector {
    private Connection connection = null;

    //Constants
    private static final String IP       = "localhost";
    private static final String PORT     = "3306";
    public static final String DATABASE  = "user_management";
    private static final String USERNAME = "usermanagement";
    private static final String PASSWORD = "1234";

    public DBConnector() throws Exception {
        Class.forName("com.mysql.jdbc.Driver").newInstance();
        String url = "jdbc:mysql://" + IP + ":" + PORT + "/" + DATABASE;
        this.connection = (Connection) DriverManager.getConnection(url, USERNAME, PASSWORD);
    }

    public Connection getConnection() {
        return this.connection;
    }
}
```

APPENDIX E – DATA ACCESS INTERFACE

```
public interface DataAccessObject {
      // Team
      public ArrayList<User> getTeamMembers(int team_id);
      public ArrayList<Team> getTeams();
      public Team getTeam(int id);
      public Team getTeam(String teamname);
      // User
      public ArrayList<User> getUsers();
      public User getUser(int id);
      public User getUser(String username);
}
```

APPENDIX F – JUNIT

```
public class DataAccessObjectTest {
    private static DBConnector connector;
    private static DataAccessObjectInterface dao;

    // Setup
    @BeforeClass
    public static void setUpClass() {
        try {
            connector = new DBConnector();
        } catch (Exception ex) {
            fail();
        }
    }

    @Before
    public void setUp() {
        dao = new DataAccessObjectImpl(connector);
    }

    // Test teams
    @Test
    public void testGetTeamMembers() {
        // Positive test
        User user;

        ArrayList<User> teamMembers = dao.getTeamMembers(1);
        assertNotNull(teamMembers);
        assertFalse(teamMembers.isEmpty());
        assertEquals(teamMembers.size(), 3);

        user = teamMembers.get(0);
        assertEquals(user.getId(), 2);
        assertEquals(user.getUsername(), "Mickey Mouse");
        assertEquals(user.getPassword(), "5678");
        assertEquals(user.isAdmin(), true);

        user = teamMembers.get(1);
        assertEquals(user.getId(), 3);
        assertEquals(user.getUsername(), "Fedtmule");
        assertEquals(user.getPassword(), "1234");
        assertEquals(user.isAdmin(), false);

        user = teamMembers.get(2);
        assertEquals(user.getId(), 7);
        assertEquals(user.getUsername(), "Pluto");
        assertEquals(user.getPassword(), "1234");
        assertEquals(user.isAdmin(), false);
    }

    @Test
    public void testGetTeamMembersInValidTeamID() {
        // Negative test
        ArrayList<User> teamMembers = dao.getTeamMembers(99);
        assertNotNull(teamMembers);
        assertTrue(teamMembers.isEmpty());
    }
```

```
    @Test
    public void testGetAllTeams() {
        ArrayList<Team> teams = dao.getTeams();
        assertNotNull(teams);
        assertFalse(teams.isEmpty());
        Team team = teams.get(0);
        assertEquals(team.getName(), "A");
        assertEquals(team.getMembers().size(), 3);
    }

    @Test
    public void testGetTeamByID() {
        // positive test
        Team team = dao.getTeam(1);
        assertNotNull(team);
        assertEquals(team.getName(), "A");
        assertEquals(team.getMembers().size(), 3);
    }

    @Test
    public void testGetTeamByInvalidID() {
        // Negative test
        Team team = dao.getTeam(99);
        assertNull(team);
    }

    @Test
    public void testGetTeamByTeamName() {
        // positive test
        Team team = dao.getTeam("A");
        assertNotNull(team);
        assertEquals(team.getId(), 1);
        assertEquals(team.getMembers().size(), 3);
    }

    @Test
    public void testGetTeamByInvalidTeamName() {
        // negative test
        Team team = dao.getTeam("Not a team name!");
        assertNull(team);
    }

    // Test users
    @Test
    public void testGetAllUsers() {
        fail();
    }

    @Test
    public void testGetUserByID() {
        fail();
    }

    @Test
    public void testGetUserByName() {
        fail();
    }
}
```