01 JDBC

* JDBC is a standard Java API for database-independent   
  connectivity
* Includes APIs for:
  + Making a connection to a database
  + Creating and executing **SQL** queries in the database
* Viewing & Modifying the resulting records
* JDBC **API** – provides the connection between the application and the driver manager
* JDBC **Driver Manager** – establishes the connection with the   
  correct driver
  + Supports multiple drivers connected to different types of   
    databases
* JDBC **Driver** - handles the communications with the database
* JDBC **API** provides several interfaces and classes:
  + **DriverManager** – matches requests from the application with the proper DB driver
  + **Driver** – handles the communication with the DB server
  + **Connection** – all methods for contacting a database
  + **Statement** – methods and properties that enable you to send SQL
  + **ResultSet** – retrieved data (set of table rows)
  + **SQLException**
* **ResultSet** maintains a **cursor** pointing to its **current row of data**
  + Not updatable
  + Iterable only once and only from the first row to the last row
  + Provides getter methods for retrieving column values from the current row
  + Retrieved information is reached by getter methods:
  + E.g.:
  + getString("column\_name")
  + getDouble("column\_name")
  + getBoolean("column\_name") etc.
  + The driver converts the underlying data to the Java type
  + The java.sql package provides all previously mentioned JDBC   
    classes
  + In order to work with JDBC we need to download a MySQL Driver – Connector/J
  + Connection with the database is established via **connection**   
    **string**
  + jdbc:<driver protocol>:<connection details> - jdbc:mysql//localhost:3306/my\_db
  + E.g. connection from previous demo:

**Connection c = DriverManager.getConnection("jdbc:mysql://localhost:3306/soft\_uni", props);**

* The JDBC **Statement interface** defines the methods and properties that enable you to send SQL commands to the database.

|  |  |
| --- | --- |
| **Interfaces** | **Recommended use** |
| Statement | For general-purpose access to your database and static SQL statements at runtime. Cannot accept parameters. |
| PreparedStatement | For SQL statements used many times. Accepts parameters. |
| CallableStatement | Used for stored procedures. Accepts parameters. |

**What is ORM?**

* **Technique** for **converting data** betweenincompatible type systems using **object-oriented programming** languages
* **Object-Relational Mapping** (ORM) allows   
  manipulating databases **using common classes and   
  objects**
  + **Java/C#/etc. classes** 🡺 **Database Tables**
  + **Database Tables** 🡺 **Java/C#/etc. classes**

**Why do we need ORM?**

* In OOP, data-management tasks act on **objects** that are almost always **non-scalar** values
* Many **database** can only store and manipulate **scalar** values, organized within **tables**
* We must **manually** convert values into groups of simpler values to store in DB and convert them back when we retrieve data

**JDBC vs ORM**

* The main difference, between JDBC and ORM, is **complexity**
* **JDBC/SQL** - If the application is simple as to present data directly from the database
* **ORM** - If the application is domain driven and the relations among objects is complex

**ORM Frameworks: Features**

* **ORM frameworks** typically **provide** the following functionality:
  + **Automatically generate SQL** to perform data operations as persist, update, delete, merge, createQuery and so on.
  + **Object model from database schema** (DB First model)
  + **Database schema from object model** (Code First model)
  + We can use and specific ORM Query Language as **HQL** or **SQL**

**ORM Advantages**

* **Productivity:** Eliminates repetitive code, Generates database automatically
* **Maintainability:** Fewer lines of code, Easier to manage object model changes
* **Performance:** Lazy loading, Caching
* **Database vendor independence** The database is abstracted, Can be configured outside the application

**ORM Disadvantages**

* **Reduced performance:** Due to overhead or auto generated SQL
* **Reduces flexibility:** Some operations are hard to implement
* **Lose understanding:**  What the code is actually doing - the developer is more in control using SQL