

Object-Relation Mapping

Hogeschool Rotterdam
Rotterdam, Netherlands

Lecture topics

- Recap Java Database Connectivity
- Object-Relation Mapping.
- Java Persistence API
- Setting up JPA with EclipseLink.

JDBC

- Provides a set of Java API for accessing the relational databases from Java program.
- It allows querying/updating database data
- JDBC represents statements using one of the following classes:

JDBC

- Provides a set of Java API for accessing the relational databases from Java program.
- It allows querying/updating database data
- JDBC represents statements using one of the following classes:
 - ① Statement – the statement is sent to the database server each and every time.

JDBC

- Provides a set of Java API for accessing the relational databases from Java program.
- It allows querying/updating database data
- JDBC represents statements using one of the following classes:
 - 1 Statement – the statement is sent to the database server each and every time.
 - 2 PreparedStatement – the statement is cached and then the execution path is pre-determined on the database server allowing it to be executed multiple times in an efficient manner.

JDBC

- Provides a set of Java API for accessing the relational databases from Java program.
- It allows querying/updating database data
- JDBC represents statements using one of the following classes:
 - 1 Statement – the statement is sent to the database server each and every time.
 - 2 PreparedStatement – the statement is cached and then the execution path is pre-determined on the database server allowing it to be executed multiple times in an efficient manner.
 - 3 CallableStatement – used for executing stored procedures on the database.

```
..
try (Connection conn = DriverManager.getConnection(
    "jdbc:somejdbcvendor:other_data_needed
    by_some_jdbc_vendor",
    "myLogin",
    "myPassword" ) ) {

    Statement stmt = conn.createStatement()
    stmt.executeUpdate( "INSERT INTO Ships(
        name) VALUE( 'destroyer_XYZ' )" );
}

..
```

Disadvantages of JDBC

- Less maintainable code for large projects
- Queries are DBMS specific

ORM

- It's a technique to map object state to the database columns

ORM

- It's a technique to map object state to the database columns
- Hides details of SQL queries from OO logic.

ORM

- It's a technique to map object state to the database columns
- Hides details of SQL queries from OO logic.
- It provides a way for data conversion between incompatible type systems

ORM

- It's a technique to map object state to the database columns
- Hides details of SQL queries from OO logic.
- It provides a way for data conversion between incompatible type systems
- Portability: DB independent

ORM

- It's a technique to map object state to the database columns
- Hides details of SQL queries from OO logic.
- It provides a way for data conversion between incompatible type systems
- Portability: DB independent
- Performance: Object and query caching mechanism

ORM

- It's a technique to map object state to the database columns
- Hides details of SQL queries from OO logic.
- It provides a way for data conversion between incompatible type systems
- Portability: DB independent
- Performance: Object and query caching mechanism
- ORM are not only related to Java, other languages provide this technique
 - C#: Linq or Entity-Framework
 - Ruby: Active Record
 - Java Hibernate, EclipseLink

ORM and Data Persistence Strategies

- There are some strategies designed for persistence of objects
- ORM framework are based on those strategies (also called pattern) and framework specific implementations
- Only two data persistence strategies will be discussed in this lecture

ORM and Data Persistence Strategies

- There are some strategies designed for persistence of objects
- ORM framework are based on those strategies (also called pattern) and framework specific implementations
- Only two data persistence strategies will be discussed in this lecture
 - Active record
 - Data mapper

Active Record Strategy

- In this strategy there is an object for every table or view that wraps a row

Active Record Strategy

- In this strategy there is an object for every table or view that wraps a row
- Objects structures are tightly coupled to the relational schema

Active Record Strategy

- In this strategy there is an object for every table or view that wraps a row
- Objects structures are tightly coupled to the relational schema
- This object encapsulates the database access, and adds domain logic on that data

Active Record Strategy

- In this strategy there is an object for every table or view that wraps a row
- Objects structures are tightly coupled to the relational schema
- This object encapsulates the database access, and adds domain logic on that data
- So this object carries both data and behavior

Active Record Strategy

- sample structure of active record

Client
name
address
phoneNumber
email
insert()
update()
delete()
getTotalSales()
generateInvoice()

```
public void update() {  
    String statement = "update Client where id = "  
        + getId() + " set name = " + getName() +  
        ".....";  
    .....  
    DB.execute(statement);  
}
```

Active Record Strategy

- What are the limitations of this pattern regarding objects structure?

Active Record Strategy

- What are the limitations of this pattern regarding objects structure?
- Think of domain object models that are distinguish from the one of the database schema

Data Mapper Strategy

- In strategy there is a layer of mappers that moves data between objects and a database
- This layer keeps both in-memory objects and database independent from each others
- The reasons for this are:

Data Mapper Strategy

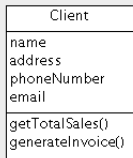
- In strategy there is a layer of mappers that moves data between objects and a database
- This layer keeps both in-memory objects and database independent from each others
- The reasons for this are:
 - Objects and relational databases have different mechanisms for structuring data

Data Mapper Strategy

- In strategy there is a layer of mappers that moves data between objects and a database
- This layer keeps both in-memory objects and database independent from each others
- The reasons for this are:
 - Objects and relational databases have different mechanisms for structuring data
 - Many parts of an object, such as collections and inheritance, aren't present in relational databases
 - Object schema and database schema do not match in many cases

Data Mapper

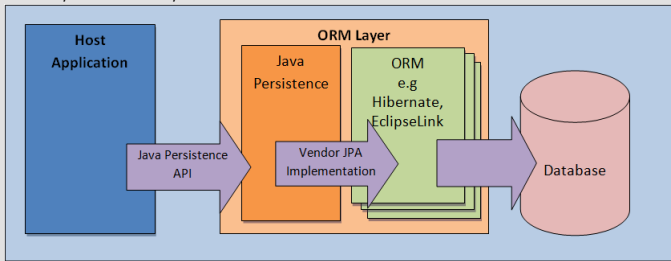
- sample structure of data mapper



```
public void update(Client c) {
    String statement = "update Client where id = "
        + c.getId() + " set name = " + c.getName() +
        " ....";
    ....
    DB.execute(statement);
}
```

Java Persistence API

- Provides the standard specification for managing the relational data in applications
- JPA is a layer between third party ORM implementations (EclipseLink or Hibernate) and the application
- It uses persistence annotations at three different levels: class, method, and field



Example of using annotation instead of mapping files

```
import javax.persistence.*;
```

```
@Entity
```

```
@Table(name = "ships")
```

```
public class Ships {
```

```
    @Id @GeneratedValue
```

```
    @Column(name = "serial")
```

```
    private int serial;
```

```
    @Column(name = "name")
```

```
    private String name;
```

```
    @Column(name = "armour")
```

```
    private String armour;
```

```
    public Employee() {}
```

```
    ...
```

Sample code for persisting a new ship in JPA with Hibernate

```
....  
Ships s = new Ships("Some_ship_name","metal");  
....  
EntityManagerFactory emf = Persistence.  
    createEntityManagerFactory("InfDev5PU");  
//create a session object in case of Hibernate  
EntityManager em = emf.createEntityManager();  
//starts a transaction  
em.getTransaction().begin();  
//calls the save method of Hibernate  
em.persist(em);  
em.getTransaction().commit();  
em.close();  
....
```

Mapping Tables and Relationships in ORM

- Mapping Simple Types like primitive java types: byte, int, short, etc.
- Mapping tables to entity classes.
- Mapping relationships:
 - In entity-classes the many-side of a relation can be mapped as Collections, Lists, Maps and Sets
 - It depends on the requirements within your application
 - Two entities cannot share a reference to the same collection instance
 - There are special annotations (ex. @OneToMany) to support mappings

Lab

- Check the tables mentioned in les 0
- Create these tables in Postgres
- Insert some new data into the database using JPA
- Print these data in your terminal using a named query