Hibernate

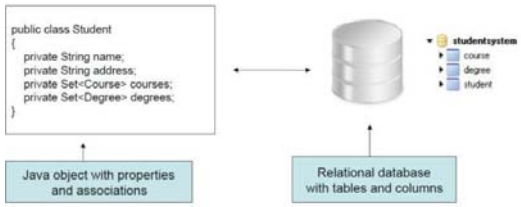
What are ORM Frameworks

- ORM: Object relational mapping
- Maps objects to relational tables
  - Written in an OO language (java, C#, …), wraps around a database
  - Makes it easy to interact with a database using OO code (as opposed to SQL)
- Helps creates consistent code (reduced SQL)
  - In practice, some specific SQL queries often needed

What is Hibernate

- Hibernate (http://hibernate.org) is an Object / Relational Mapping (ORM) framework for Java and relational databases
- Java based
- Open Source
- Can be used with other application frameworks (including Spring and Spring MVC)

Why ORM



Example

- First create database

```
create database elec5619;
create table employee(
id int,
age int,
first varchar(50),
last varchar(50),
PRIMARY KEY (id)
);
```

Example

- First create database

```
create database elec5619;
create table employee(
id int,
age int,
first varchar(50),
last varchar(50),
PRIMARY KEY (id)
);
```

Edit the database using SQL

```
mysql> use elec5619
Database changed
mysql> desc employee;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int(11)     | NO   | PRI | NULL    |       |
| age   | int(11)     | YES  |     | NULL    |       |
| first | varchar(50) | YES  |     | NULL    |       |
| last  | varchar(50) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
```

Add object using SQL

```
ysql> INSERT INTO employee (id, age, first, last) VALUES (10, 25, 'Luna', 'Girl');
uery OK, 1 row affected (0.03 sec)

ysql> select * from employee;
----+------+-------+------+
 id | age  | first | last |
----+------+-------+------+
 10 |   25 | Luna  | Girl |
----+------+-------+------+
row in set (0.00 sec)
```

Use JDBC

```
@RequestMapping(value = "/jdbcAdd", method = RequestMethod.GET)
public String jdbcAdd(Locale locale, Model model) {

    // JDBC driver name and database URL
    String JDBC_DRIVER = "com.mysql.jdbc.Driver";
    String DB_URL = "jdbc:mysql://localhost/elec5619";

    //  Database credentials
    String USER = "root"; //
    String PASS = "root";

    Connection conn = null;
    PreparedStatement preparedStatement = null;
    try{
       //Register JDBC driver
       Class.forName("com.mysql.jdbc.Driver");

       //Open a connection
       System.out.println("Connecting to database...");
       conn = DriverManager.getConnection(DB_URL,USER,PASS);
```

Use JDBC

```
conn = DriverManager.getConnection(DB_URL,USER,PASS);

String insertTableSQL = "INSERT INTO employee (id, age, first, last) VALUES (?,?,?,?)";

preparedStatement = conn.prepareStatement(insertTableSQL);
preparedStatement.setInt(1, 10);
preparedStatement.setInt(2, 25);
preparedStatement.setString(3, "Luna");
preparedStatement.setString(4, "Girl");

// execute insert SQL statement
preparedStatement.executeUpdate();
System.out.println("Record is inserted into employee table!");

//Clean-up environment
preparedStatement.close();
conn.close();
```

Using Hibernate – first define object

```
package au.edu.sydney.domain;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "person")
public class Person {

    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;

    @Column(name = "age")
    private int age;

    @Column(name = "first")
    private String first;

    @Column(name = "last")
    private String last;

    public int getId() {
        return id;
    }

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

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public String getFirst() {
        return first;
    }

    public void setFirst(String first) {
        this.first = first;
    }

    public String getLast() {
        return last;
    }

    public void setLast(String last) {
        this.last = last;
    }

}
```

Using Hibernate – then create new object.

```
@Autowired
private SessionFactory sessionFactory;

@RequestMapping(value = "/hibernateAdd", method = RequestMethod.GET)
public String hibernateAdd(Locale locale, Model model) {

    Person p = new Person();
    p.setAge(20);
    p.setFirst("FirstName");
    p.setLast("lastName");

    sessionFactory.getCurrentSession().save(p);
    return "home";
}
```

ORM translates this to SQL

Limitation

```
@RequestMapping(value = "/hibernateAdd", method = RequestMethod.GET)
public String hibernateAdd(Locale locale, Model model) {

    Person p = new Person();
    p.setAge(20);
    p.setFirst("FirstName");
    p.setLast("lastName");

    sessionFactory.getCurrentSession().save(p);
    return "home";
}
```

Limitation

```
@RequestMapping(value = "/hibernateAdd", method = RequestMethod.GET)
public String hibernateAdd(Locale locale, Model model) {

    Person p = new Person();
    p.setAge(20);
    p.setFirst("FirstName");
    p.setLast("lastName");

    sessionFactory.getCurrentSession().save(p);
    return "home";
}
```

Hibernate specific

Data access object

- Separate parts of an application where possible, especially if they may evolve independently
- Details of low-level data access should be hidden from high-level business services

Using a Data Access Object (DAO)

```
package au.edu.sydney.dao;

import javax.annotation.Resource;

import org.hibernate.SessionFactory;
import org.springframework.stereotype.Repository;

import au.edu.sydney.domain.Person;

@Repository(value = "personDao")
public class PersonDao {

    @Resource
    private SessionFactory sessionFactory;

    public SessionFactory getSessionFactory() {
        return sessionFactory;
    }

    public void setSessionFactory(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
    }

    public void savePerson(Person person) {
        sessionFactory.getCurrentSession().save(person);
    }
}
```

Using a Data Access Object (DAO)

```
@Autowired
private PersonDao personDao;

@RequestMapping(value = "/hibernateDaoAdd", method = RequestMethod.GET)
public String hibernateDaoAdd(Locale locale, Model model) {
    Person p = new Person();
    p.setAge(20);
    p.setFirst("FirstName");
    p.setLast("lastName");

    personDao.savePerson(p);

    return "home";
}
```

Adding further functionality

- Additional functionality should be added using a 'service' layer
- Service layer makes use of DAO

```java
package au.edu.sydney.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import au.edu.sydney.dao.PersonDao;
import au.edu.sydney.domain.Person;

@Service(value="personService")
// @Transactional
public class PersonService {

    @Autowired
    private PersonDao personDao;

    // business logic of registering a Person into the database
    public void registerPerson(Person person) {

        // Step 1: check whether this person is already in the database

        // Step 2: if not, save this person into the database
        personDao.savePerson(person);
    }

}
```

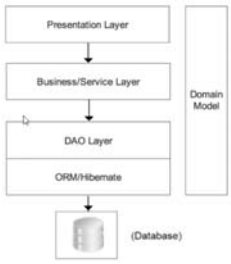
```java
@Autowired
private PersonService personService;

@RequestMapping(value = "/hibernateDaoServiceAdd", method = RequestMethod.GET)
public String hibernateDaoServiceAdd(Locale locale, Model model) {
    Person p = new Person();
    p.setAge(20);
    p.setFirst("FirstName");
    p.setLast("lastName");

    personService.registerPerson(p);

    return "home";
}
```

System Architecture



Queries

- Fetch by ID
- HQL (Hibernate Query Language)
- Hibernate Criteria API
- Native SQL queries

Fetch by ID

- Identify a single instance of a class using student ID

```java
Student me = (Student) session.get(Student.class, myStudentId);
```

- Returns either the instance or null

```java
Student me = (Student) session.load(Student.class, myStudentId);
```

- Returns either the instance, or throws an exception

Hibernate Query Language

- Object oriented form of SQL

```java
Query q = session.createQuery("from Student s where s.name = :sname");
q.setString("sname", "Yu Zhao");
List result = q.list();
```

Hibernate Criteria API

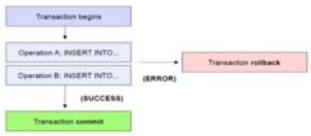
- Create a query by manipulating a criteria instance at runtime

```
Criteria criteria = session.createCriteria(Student.class);
criteria.add(Expression.like("name", "Yu Zhao"));
List result = criteria.list();
```

```
Student me = new Student();
me.setName("Yu Zhao");
Criteria criteria = session.createCriteria(Student.class);
criteria.add(Example.create(me));
List result = criteria.list();
```

Transaction

- Ensure database operations either execute entirely, or not at all
- Add @transactional



Useful resources

- Bauer, C., & King, G. (2005). Hibernate in action.
- Hibernate Documentation (http://hibernate.org/orm/documentation/)
- Source codes https://github.com/ianliu0420/elec5619_demos