COS30041 Create Secure and Scalable Software

Lecture 03b Java Persistence API (JPA)

Learning Objectives

- After studying the lecture material, you will be able to
  - Understand and describe what an entity class is
  - Understand and describe the features that Java Persistence API has to offer
  - Understand the issues involved in programming entity class using Java Persistence API
  - Program entity class
  - Program client applications that call the services provided by entity class

Pre-requisite

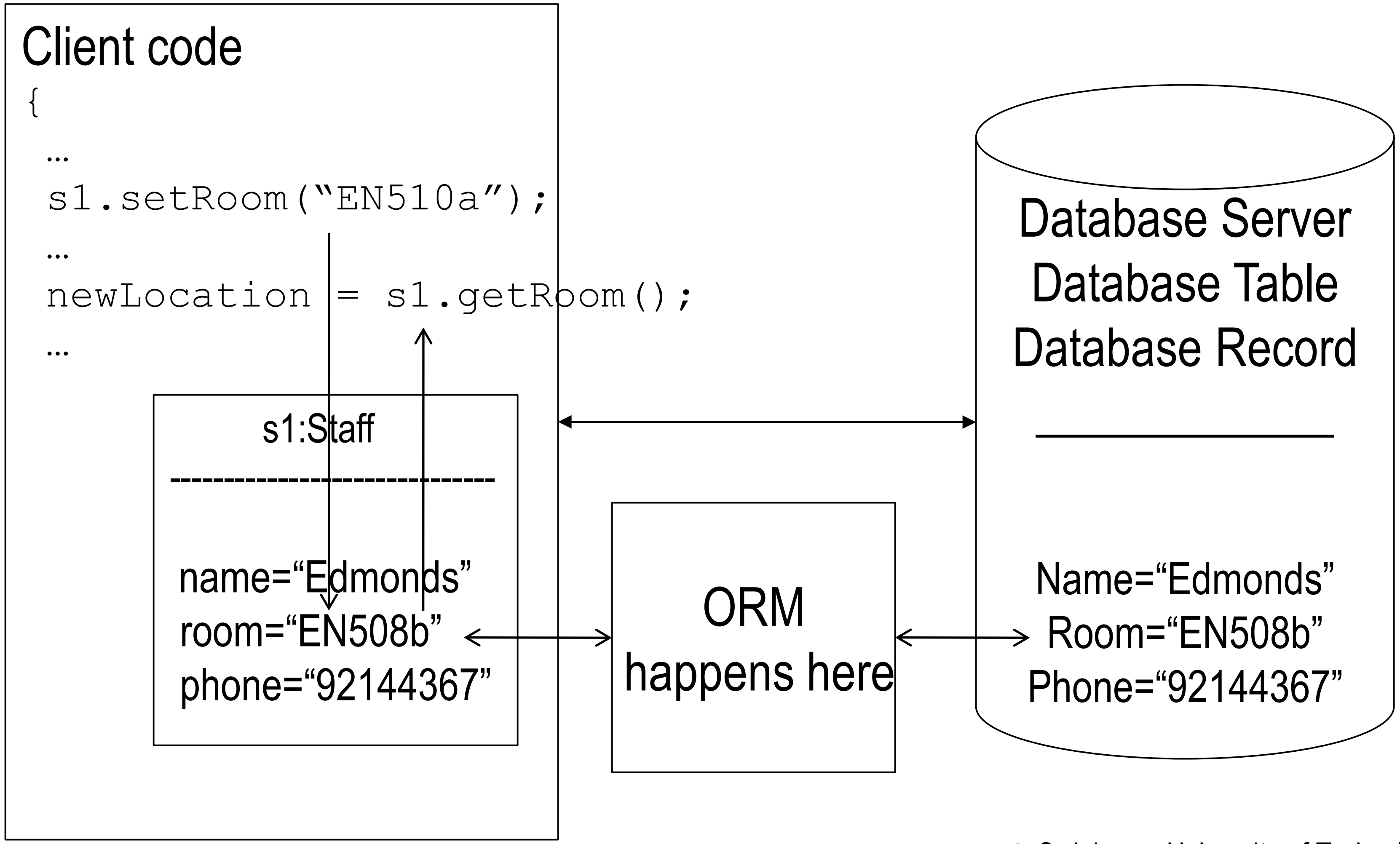
- Database concepts
- Writing simple SQL statements
- JDBC, some knowledge would be good

Outline

- Entity Class
- The Primary Key
- Programming a JPA application
  - Program the Entity Class
  - Use the EntityManager API
- Programming Example

Object-relational Mapping, ORM

- A mapping between objects (in OOP) and database records



s1: Staff (Class)

----

name="Edmonds"
room="EN508b"
phone="92144367"

----

s1 is an object

ORM
JPA:

```
        em.persist(s1);
```

Use the EntityManager API:

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("EX-EntityPU");
EntityManager em = emf.createEntityManager();
```

Roadmap

- **Entity Class**
- The Primary Key
- Programming a JPA application
  - Program the Entity Class
  - Use the EntityManager API
- Programming Example

Entity Class

- [MEJB3] A Plain Old Java Object (POJO) that is used to persist objects that can be stored in permanent storage
  - Databases or Legacy systems
- [JEE7T;37-1] A lightweight persistence domain object
  - Typically, it represents a table in a relational database (RDB)
  - Each entity instance corresponds to a row in the table
- Example
  - Bank Account
  - Customer Data

Why Entity Class?

- Is handy to treat data as objects
- Can associate simple methods with objects
- Can store the data in memory for performance
- Using POJO can achieve much simpler code
  - cf those in the EJB's Entity Bean (old Java EE 1.4 or earlier)
- Detached from EJB allowing it to exist in Application Client, hence no need to have separate DTOs
  - Changing attributes in entity class on the client side won't affect the "same" copy on the server, and hence will not affect those in the databases [Note: ORM only occurs on the server]
  - Easy to cause confusion, though!

Terminologies

- Entity Class Instance (usually on the server)
  - The in-memory view of the database (as an instance of the entity class)
- Entity Data
  - The physical set of data stored in the database
- Example
  - Entity Class “Account” models the database table “Account”
  - The “Account” entity class instance stores a particular “account” in the database
  - Assume “Account” has 2 instance variables “accountId” and “accountBalance”, each refers to a particular field in the database table

Entity Class Composition

The **data** that it represents

- All fields need to be serializable
- The Primary Key
- Map to an entity definition of a database schema using features provided by Java Persistence API

Roadmap

- ■ Entity Class
- ■ **The Primary Key**
- ■ Programming a JPA application
  - ☐ Program the Entity Class
  - ☐ Use the EntityManager API
- ■ Programming Example

For details, please see [JEE7T,p.37-6]

The Primary Key

- The unique identifier of the entity data instance
  - May contain any number of attributes
  - Simple vs Composite
- Simple Primary Key
  - Use this when the primary key is just one data field
  - Use an instance variable in the Entity Class
  - Must be serializable
  - Use "`@Id`" annotation to indicate it is a simple primary key
- Composite Primary Key defined in Primary Key Class

For details, please see [JEE7T,p.37-6]

The Primary Key (cont'd)

- The Primary Key Class
  - Use a serializable object, if the primary key has multiple data fields
  - **Must be public**
  - Have a public default constructor
  - **Must implement** the hashCode() and equals(Object other) methods
  - Must be serializable
  - Composite primary key must be represented and mapped to multiple fields of the entity class (corresponding names and types must match)

Roadmap

- Entity Class
- The Primary Key
- **Programming a JPA application**
  - Program the Entity Class
  - Use the EntityManager API
- Programming Example

Programming a JPA application

- Program the Entity Class
- Use the EntityManager API
- Program the client

Program the Entity Class – Requirements

For details, see [JEE7T,p.37-1]

- Must be annotated with `@Entity`
  - `javax.persistence.Entity`
- Must have a public or protected, no-argument constructor
  - May have other constructors
- Must not be declared final
  - Not in the methods or persistent instance variables
- Must implement the `Serializable` interface
- Persistent instance variables
  - Must be declared private, protected, or package-private
  - Can only be accessed directly by the entity class's methods

Program the Entity Class – Req. (cont'd)

- *Persistent Field – persistence on instance variable
- Persistent Property – persistence on getter and setter
- Both fields and properties require
  - ☐ Java primitive data types
  - ☐ `java.lang.String`
  - ☐ Other serializable types including
    - ☐ Wrappers of Java primitive types, `java.math.BigInteger`, …
  - ☐ Enumerated types
  - ☐ Other entities and/or collections of entities
  - ☐ …

For details, please see [JEE7T,p.37-1]

*Note: The approach in this subject.

Program the Entity Class – JPA

- [NetBeans 8.2] A normal Java POJO project
  - **The Entity Class**
  - **The Primary Key Class** (if needed)
  - **The User-defined Exception Class** (Optional)
  - The Persistence Descriptor – defines the database connections
    - persistence.xml
  - The Vendor-Specific Files

Program the Entity Class – JPA

- Naming Convention
  - The Entity Class – Class (e.g. Employee)
  - The Primary Key Class – ClassKey (e.g. EmployeePK)

Roadmap

- Entity Class
- The Primary Key
- Programming a JPA application
  - Program the Entity Class
  - **Use the EntityManager API**
- Programming Example

Use the Entity Manager API – JPA

- The EntityManager Class is to perform all ORM related operations on the entity class
- It is responsible for
  - Accessing the entity class data
  - Managing the persistence of the entity class data

Using the EntityManager API – JPA (cont'd)

- Entity lookup and queries
    - Search and return the required database records to the applications
- Database synchronization operations
    - Update the relevant data
- Entity life-cycle management
    - Manage a limited number of entity class instances for probably the entire database

Use the EntityManager API – JPA (cont'd)

- Java uses the famous "Factory Design Pattern" to create a reference to an entity manager object
- Steps:
  - Use database connection information stored in persistence.xml file to get an EntityManagerFactory object

    -
      ```
      EntityManagerFactory emf =
      Persistence.createEntityManagerFactory("EX-
      EntityPU");
      ```
  - Use the factory object to create an entityManager object

    -
      ```
      EntityManager em = emf.createEntityManager();
      ```

Use the EntityManager API – JPA (cont'd)

- Once the entity manager object is obtained, use it to perform the following
  - Create a database record – `em.persist()`
  - Update a database record – `em.merge()`
  - Delete a database record – `em.remove()`
  - Search a database record – `em.find()`
- Use the Transaction API to manage the transaction
  - Begin a transaction – use `transaction.begin()`
  - Commit a transaction – use `transaction.commit()`

Using the EntityManager API – Lookup and Query

- Use the EntityManager to get an instance of `javax.Persistence.Query` with specific query statement
- Execute the query: many possibilities – one here
  - *`createQuery(String qlString)`
- Example: get all records from the Account Table

```
Query query =
    entityManager.createQuery(
        “SELECT a FROM Account a”);
return (List<Account>) query.getResultList();
```

*Note: We will only use JPQL query for database independent purposes

Using the EntityManager API – Lookup and Query (cont'd)

- Execute the query: other possibilities
- *`createNamedQuery(String queryName)`
  - Need to define the query name using `@NamedQuery` annotation in the Entity Class (not the EntityManager class)
  - *`@NamedQuery(name = queryName, queryString = qlString);` or
  - `@NamedQuery(name = queryName, queryString = sqlString)`
- `createNativeQuery(String sqlString)`

*Note: We will only use JPQL query for database independent purposes

Using the EntityManager API – Synchronization

- Synchronization is “automatic” after calling `persist()`
- “Manual” synchronization is possible
  - `setFlushMode()`
    - COMMIT – synchronization occurs at commit time
    - AUTO – synchronization occurs at commit time **and** before query execution
  - `flush():` Enforce synchronization of all entities in the persistence context
  - `refresh():` Refresh the entity instance from the database

Using the EntityManager API – Life Cycle

- `persist()`
  - Hand over the entity instance to the `EntityManager` class to manage the persistence
  - Any changes of the attributes in the entity instance will be reflected in the corresponding database record

Roadmap

- Entity Class
- The Primary Key
- Programming a JPA application
  - Program the Entity Class
  - Use the EntityManager API
- **Programming Example**

Programming Example – EX-JPA-Customer

- WANT: An Entity Class "`Customer`" to allow a customer to deposit money to and withdraw money from their own accounts
- The Entity Class – `Customer.java`
- The EntityManager helper – `CustomerEM.java`
- The Client – `CustomerApp.java`
- Download `EX-JPA-Customer.zip` from Canvas

https://www.tutorialspoint.com/jpa/jpa_entity_managers.htm

References

- [MEJB3] R.P. Sriganesh, G. Brose, M. Silverman (2008) *Mastering Enterprise JavaBeans 3.0*, 4th ed., John Wiley & Sons
  - Chapter 6
- [JEE7T] E. Jendrock et al. (2014) *The Java EE 7 Tutorial*, Oracle, Sep 2014
  - Chapters 37 – 39