Java Persistence API

Instructor and Lab Guide

Chapter 1

- **Lecture**
  - pages 3-8, 11-15, 30

- **Hands On**
  - Create a JPA Working Set
  - Create a new JPA Project
    - Disable Library Configuration on third page of the wizard.
  - Convert to Maven Project
  - Add dependencies for:
    - mysql-connector-java
    - hibernate-entitymanager
    - org.eclipse.persistence.jpa

  - Build a Job entity (id, name, minimumSalary, maximumSalary)
  - Create a JUnit test for the Job

- **Labs**
  - Build a Location entity. Include fields for id, and squareFootage. Use the Job entity as a guide.

  - Create a JUnit test for Location. Use the Job test as a guide.

  - Build an Employee entity. Include fields for id, firstName, middleName, lastName, gender, salary, and commissionPct.

  - Create a JUnit test for Employee.

  - Build a Department entity. Include fields for id and name.

  - Create a JUnit test for Department.

Chapter 2A - Annotations

- **Lecture**
  - pages 41,43-44, 46-49
- **Hands On**
  - Add @GeneratedValue(strategy=GenerationType.IDENTITY) to Job, Location, Employee, and Department.

- **Lecture**
  - pages 50-52
- **Hands On**
  - Add hiredate to Employee with the @Temporal annotation.

- **Lecture**
  - page 53
- **Hands On**
  - Add an enum called Gender, with values M, F. Modify Employee to use the Gender enum with the @Enumerated annotation
  - Add an enum called State, with (minimally) values GA, MA, NC, NJ

- **Lecture**
  - pages 54-56
- **Hands On**
  - Add the Address as an @Embeddable type with the streetAddress annotated with @Column(name="street_address")
  - Add an address field to Location with the @Embedded annotation
  - Add an address field to Employee with the @Embedded and @AttributeOverride like p. 56

- **Lecture**
  - page 57

- **Labs**
  - Add the locationId into Department. Write a program that does a find to get a Department and another find to get a Location based on the id from Dept.

Chapter 2B - Basic Queries

- **Lecture**
  - Teach them find all:
    - List<Department> depts = em.createQuery("SELECT d FROM Department d", Department.class).getResultList();

- **Labs**
  - Create a program that lists the first name and last name of each employee in the database.

  - Create a program that lists the city and state for each location in the database.

  - Create a program that lists the name of each department as well as the city and state that the department is a member of.

Chapter 2C - Relationships and Many To One

- **Lecture**
  - pages 59-65
- **Hands On**
  - Add @ManyToOne and @JoinColumn to Department's location field. Add @OneToMany with mappedBy to Location's List<Department> field.
  - Add JUnit to test the above relationship.
- **Demo**
  - Instructor creates a program that lists all of the locations and the number of departments assigned to each.

- **Lecture**
  - page 69
- **Demo**
  - Instructor will modify Department and Location to be Unidirectional

- **Labs**
  - Create a OneToMany Bidirectional relationship between Job and Employee.

  - Write a JUnit test for this relationship.

  - Create a program that lists all of the jobs in the database as well as how many employees are assigned to each job.

  - Create a OneToMany Bidirectional relationship between Department and Employee.

  - Write a JUnit test for this relationship.

  - Create a program that lists all of the departments in the database as well as how many employees are assigned to each department.

Chapter 2D - OneToOne

- **Lecture**
  - pages 70-71

- **Hands On**
  - Modify Department add @OneToOne and @JoinColumn on private Employee manager, add @OneToOne with mappedBy to private Department deptManaged.
  - Add JUnit to test above relationship.

- **Labs**
  - Create a program that lists the name of each department and the first and last name of the the employee who manages it.

  - In preparation for the next section, create two new classes: Project (id, name, startDate, endDate) and Assignment (id, startDate, endDate).

Chapter 2E - ManyToMany

- **Lecture**
  - pages 74-78

- **Hands On**
  - Modify Employee and Project adding the ManyToMany relationship.

- **Labs**
  - Build a program that lists all of projects that Employee 1066 has participated in and the start and end date of the Project.

  - Build a ManyToOne between Assignment and Employee.

  - Build a ManyToOne between Assignment and Project.

  - Build a program that lists all of the assignments that Employee 1066 has participated in and the start and end date of the Assignment. Make sure to display the project name that you can retrieve from the assignment's relationship with the project.

Chapter 2F - Collections

- **Lecture**
  - page 80

- **Demo**
  - Add @OrderBy("salary") to the Department employees field. Create a program that lists all employees in department 6 sorted by salary, then switch to DESC.

- **Lecture**
  - pages 81-82

- **Hands On**
  - Add the addEmployee() and removeEmployee() methods into Department.

```
public void addEmployee(Employee employee) {
       if (!employees.contains(employee)) {
              // add employee to this department's list
              employees.add(employee);
              // remove employee from their old department
              if (employee.getDepartment() != null) {
                     employee.getDepartment().getEmployees().remove(employee);
              }
              // Bidirectional, so add department to the employee
              employee.setDepartment(this);
       }
}
public void removeEmployee(Employee employee) {
       if (employees.contains(employee))  {
              // remove employee from this department's list
              employees.remove(employee);
              // Bidirectional, so null out the department in the employee
              employee.setDepartment(null);
       }
}
```

- **Labs (Optional)**
  - Add methods to add and remove a Department from Location.

- **Lecture**
  - page 83

Chapter 3A - Entity Manager

- **Lecture**
  - pages 95-102
- **Demo**
  - Instructor will create an application that uses find to locate a Job (7 - HR), then call a set method to update it (Human Resources).
  - Instructor will create an application that uses persist to add a new Job to the database, calling em.contains() to verify that it is managed.
  - Instructor will create an application that uses find, then remove to take it back out.
  - Instructor will create an application that uses find, then detach, then merge calling em.contains() throughout. Stress that merge() returns merged object.
- **Lecture**
  - pages 103 - 107
- **Labs**
  - Create an application that uses find to locate a Location (1 - Atlanta), then call a set method to update the square footage. Verify in the database that the change took effect.
  - Create an application that uses persist to add a new Location to the database, make sure to set the square footage and address. Don't worry about the departments.
  - Create an application that uses find to locate the previous Location you just added, then remove it.
  - Create an application that retrieves Department 1 using em.find(). Create a new Job using values of your choosing. Create a new

Employee, setting all of the non-null values: first name, last name, department, and job. For department use the one you just found. For Job, use the one you just created. Persist the employee and job to the database in that order (employee, then job). Notice the error. Change the persist order to job, then employee. Why does it work now?

- Write another program to remove the job and employee you just created. Does order matter in the remove calls? (YES, remove employee, before job)

Chapter 3B - Queries

- **Demo**
  - Remind students of Find All Query:

```
List<Department> depts =
     em.createQuery("SELECT d FROM Department d",
     Department.class).getResultList();
```

  - Show students a Query with a WHERE clause:

```
List<Department> depts = em.createQuery(
     "SELECT d FROM Department d WHERE d.location.id IN (1,2)",
     Department.class).getResultList();
```

  - Show students a Query with a WHERE clause that has a single result:

```
Department department = em.createQuery(
     "SELECT d FROM Department d WHERE d.id = 7",
     Department.class).getSingleResult();
```

  - Show students a Query with a WHERE clause that has a query parameter:

```
Department department = em.createQuery(
     "SELECT d FROM Department d WHERE d.id = :id",
     Department.class).setParameter("id", 7).getSingleResult();
```

- **Labs**
  - Create a program that loops through all of the employees in the database whose salaries are outside the min/max as prescribed by their job. Update each employee’s salary which is is below the min to the min.

Chapter 4 - JPQL

- **Lecture**
  - pages 129-130, 135-160

- **Demos Throughout**
  - Make sure students understand return types from queries.
  - Java Code Using JPQL that returns and Integer (location.squareFootage), an Object (location.address), and Object[] (location.address.city, location.squareFootage) in the results:

```
List<Integer> results = em.createQuery(
     "SELECT l.square_footage FROM Location l",
     Integer.class)
     .getResultList();

List<Address> results = em.createQuery(
     "SELECT l.address FROM Location l",
     Address.class)
     .getResultList();
List<Object[]> results = em.createQuery(
     "SELECT l.address.city, l.squareFootage FROM Location l",
     Object[].class)
     .getResultList();
```

- **Labs**
  - Create an application that prints out the square footage of each location.

  - Create an application that lists each employee's first name, last name, salary, and commission percentage. Also list what a hypothetical bonus might be if it was based on the salary times the commission percentage.

  - Create an application that lists each employee's first name, last name, salary, job minimum and job maximum for each employee whose salary is above the job maximum or below the job minimum. Write this query from the Employee's point of view (the from clause will use Employee). Make sure to use the between operator in your where clause.

    - Rewrite the previous application, this time from the Job's point of view (the from clause will use Job).

    - Create an application that lists all of the project names on which each employee is currently working. Assume that currently means a null end date for their Assignment. In addition to printing employee names and project names, also include the assignment start dates.

- **Independent Study**
    - pages 163-169