Work through the following materials this week:

1. Read specified sections of [Java Persistence](https://en.wikibooks.org/wiki/Java_Persistence) and explain the given concepts.
   1. Chapter 3, “What is Java persistence?”
      1. *Persistence* in Java and how it relates to the *impedance mismatch* discussed in unit 8

Persistence describes data that outlived the process to create it.

Relational DBs as the standard form of persistence storage.

Impedance mismatch are the differences that occur between the database model and the programming language model.

Thus, the java persistence API helps avoid impedance mismatch by describing the management of relational data in applications using the java platform.

* + 1. *POJO*s

Are *Plain Old Javascript Objects.* It is an ordinary Java object not bound by any special restriction and not requiring any class path.

* 1. Chapter 4, “Persistence Products, Which to Use?” — Name 2–3 of the major JPA implementations and indicate which one we are using in this class.

EclipseLink (the one we are using): supports a number of persistence standards including JPA, JAXB, JCA, SBO. Based on the TopLink product.

TopLink: built by oracle. Bundles open source EclipseLink for a lot of its functionality. Includes sophisticated caching and performance features.

* 1. Chapter 5, “Mapping, Round Pegs into Square Holes” — Annotate this Java class for object-relational mapping using *Java Annotations* with the property *access type*.
  2. @Entity
  3. public class Solder {
  4. @Id
  5. @GeneratedValue
  6. private String name;
  7. @Column(name=”RANK”)
  8. private String rank;
  9. @Column(name=”SERIAL\_NUMBER”)
  10. private long serialNumber;
  11. @ManyToOne
  12. Private Soldier commander;
  13. @ManyToMany
  14. Private List<Battle> battles;
  15. public String getName() { return name; }
  16. public void setName(String name) { this.name = name; }
  17. public String getRank() { return rank; }
  18. public void setRank(String rank) { this.rank = rank; }
  19. public String getSerialNumber() { return serialNumber; }
  20. public void setSerialNumber(String serialNumber) { this.serialNumber = serialNumber; }

}

In particular, add annotations for the following.

* + 1. *Persistence Entity* — Annotate the class as a persistence entity (Chapter 5 overview). ok
    2. *OID* — Annotate the object ID (Section 5.2). ok
    3. *Basic* fields — Annotate the non-ID fields (Section 5.6). ok
    4. *Relationships* — Add new fields/accessors/annotations that exemplify the following JPA relationships (Section 5.7 overview & Sections 5.7.2–5.7.4).
       - A *many-to-one* relationship to a commander — The commander is another Solder object. ok
       - A *many-to-many* relationship to a Battle class — You don’t need to add the Battle class. ok

You don’t need to run this code, but be familiar with the concepts and the annotations that implement them.

* 1. Chapter 6, “Runtime, Doing the Hokey Pokey (EntityManager)”
     1. *Entity Manager* (Chapter 6 overview)

Is the main runtime class – accessing & processing objects from the application.

It is an object-oriented API so does not map directly to database SQL/DBM operations.

* + 1. *Injection* — We’ll use J2EE dependency injection to access to a container-managed EntityManager (Chapter 6 overview).

It is possible for the Entity Manager to be injected into the Session Bean @PersistenceContext or @PersistenceUnit

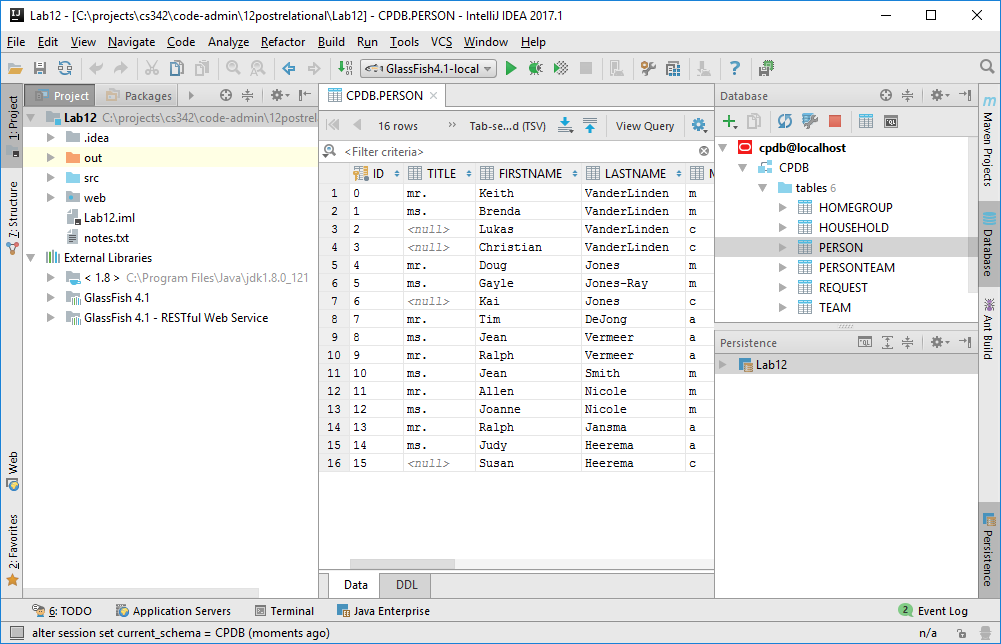
* + 1. *Persistence operations* — Name the four basic persistence operations (Section 6.2).

Persist (insert)

Merge(update)

Remove(delete)

Find(select)

[](https://cs.calvin.edu/courses/cs/342/kvlinden/12postrelational/images/guide12.png)

1. Review Intellij’s support to database connections and JPA.
   1. Create a new J2EE application as you created the JDK/J2EE application in the previous lab, activating the following “Java Enterprise” libraries/frameworks.
      1. Web Application 3.1
      2. J2EE Persistence 2.1; Provider: EclipseLink; setting up the library later
      3. RESTful Web Service; using the library from the GlassFish installation

Store it in your repo under cs342/12postrelational/Lab12.

* 1. [Databases and SQL](https://www.jetbrains.com/help/idea/2016.3/databases-and-sql.html) — Note that Oracle must be running with the CPDB database loaded for this data connection to work.
     1. Open the Intellij database tool pane by choosing *View→Tool Windows→Database*.
     2. Create a new data source for your existing Oracle CPDB database, name itcpdb&localhost and make sure that “Test Connection” works properly. You may need to download a required library (to make IDEA happy; GlassFish already has all the required libraries).
     3. Fetch the records in the Person table by double-clicking on the table name in the Database pane.
  2. [Java Persistence API (JPA)](https://www.jetbrains.com/help/idea/2016.3/java-persistence-api-jpa.html)
     1. Open the Intellij “Persistence” tool pane by choosing *View→Tool Windows→Persistence*.

Store a (low-resolution) screen dump image of your new app in the IDE. It should look something like the one shown to the right (but be lower-resolution). You’ll use this app as the basis for lab 12.

$Date: Fri, 14 Apr 2017 12:45:30 -0400 $