# Java Persistence API

## Overview

In this workshop we are going to use JPA by

1. Adding persistence.xml file and configure it.
2. Configuring EntityManager in the main method.
3. Creating objects and adding to the tables.
4. Running some queries.
5. Removing data.

## Using JPA in our project

*This workshop is based on the* ***One to Many relation between Tutor and Student class.***
So if you have not done the one to many relation workshop do it first. Or you can download the latest project that we have done here.

Now we want to use JPA instead of Classic Hibernate. So we need to change several things.
First we need to use a file called persistence.xml.

### Adding persistence.xml to the project and configure the file

You can download the *persistence.xml* file here.

Create a new folder under src/main/resources and call it *META-INF*
Put the *persistence.xml* in the META-INF.

Copy the driver class: org.apache… from *hibernate.cfg.xml*
and paste it in the 'value' part of this line instead of "TODO" in persistence.xml:

```
<property name="hibernate.connection.driver_class" value="!!TODO!!"/>
```

Do the same for the url in this line:

```
<property name="hibernate.connection.url" value="!!TODO!!"/>
```

So it becomes:

```
<property name="hibernate.connection.driver_class"
     value="org.apache.derby.jdbc.ClientDriver"/>
<property name="hibernate.connection.url"
```

```
       value="jdbc:derby://localhost:50000/hibernate"/>
```

And set the following:

```
    <property name="hibernate.show_sql" value="false" />
    <property name="hibernate.format_sql" value="false" />
    <property name="hibernate.hbm2ddl.auto" value="create" />
```

We have all the configuration now in this file and there is no need for hibernate.cfg. So you may delete the *hibernate.xml* file from your project.

## Configuring JPA in the main method

In the main method remove all the sessionFactory, session and transaction code and add these lines instead:

```
EntityManagerFactory emf =
             Persistence.createEntityManagerFactory("databaseConfig");
EntityManager em = emf.createEntityManager();
EntityTransaction tx = em.getTransaction();
tx.begin();
```

And at the end of the method:

```
tx.commit();
em.close();
```

These are the libraries you need to import:

```
import jakarta.persistence.EntityManager;
import jakarta.persistence.EntityManagerFactory;
import jakarta.persistence.EntityTransaction;
import jakarta.persistence.Persistence;
```

Delete the helper method which is creating a sessionfactory in this class too.

## Creating and persisting students and tutor in the main method

We create one tutor and 3 students and we persist them in the database.
And then we print all the students that are in t1's teaching group.

After `tx.begin();` add this code:

```
Tutor t1 = new Tutor("ABC123", "Teacher 1", 290000);
em.persist(t1);

Student s1 = new Student("Student1", "1-STU-2019");
t1.addStudentToTeachingGroup(s1);

Student s2 = new Student("Student2", "2-STU-2018");
t1.addStudentToTeachingGroup(s2);

Student s3 = new Student("Student3", "3-STU-2017");
t1.addStudentToTeachingGroup(s3);

em.persist(s1);
em.persist(s2);
em.persist(s3);

Set<Student>allStudents = t1.getTeachingGroup();
System.out.println(allStudents.size());
```

Run the code and check the result in the database.

## Changing update to create in the persistence.xml

We change the **'create'** to **'update'** in the *persistence file*.
Create will automatically drop the tables whenever we run the code and create new tables.
But 'update' will only update the data that we have in the tables. 'update' is good when we have data in the database and we only want to retrieve something from the table.

## Creating queries in the main method

Now we want to test some queries.
Comment out the code relating to creating a teacher and students.

And add the following code:

```
      Tutor tutor = em.find(Tutor.class, 1);
      System.out.println(tutor);
```

find method will do the same as get when we were using hibernate.

Run the code and check if you get the name of the teacher correct. (You need a toString() overridden in Tutor class).

An example of toString method:

```
public String toString() {
   return name;
}
```

Now add the following:

```
       Set<Student> students = tutor.getTeachingGroup();
        for (Student s: students) {
            System.out.println(s);
        }

        Student student =em.find(Student.class, 2);
        System.out.println(student);
```

What we are doing above is:
1.we are getting the teaching group for a specific tutor and storing them in a Set of students which we call it students. Within a ‘for each’ loop, we are printing them.

2.Then we are retrieving a student who has id=2 and we store it in the *student* and we print it out.

Run the code. We should get the name of all the students and again the student which has id 2 in the table.

## Delete data

We can delete the student with id 2, by the following command:

```
em.remove(student);
```

Run and check the result in the student table.