Using Java Persistence API for Java SE 7 Desktop applications in NetBeans 7

Purpose

This tutorial demonstrates setting up Java Persistence API for Java SE 7 desktop applications in NetBeans 7.

Time to Complete

Approximately 30 minutes.

Overview

The Java Persistence API(JPA) provides an object/relational mapping facility for managing relational data in Java applications. JP A is a lightweight, POJO-based framework for object-relational mapping. The mapping between Java objects and a relational database is done using annotations and/or XML deployment descriptors. Though JP A is a part of EJB 3 Specification it can be used in Java SE applications, outside of the Java EE environment .

In this tutorial, you will create and configure a persistent Unit using JP A. You will

- Create a database connection
- Generate Entity classes for tables in the database
- Persist data in the database using Entity classes

Software and Hardware Requirements

The following is a list of software requirements:

- Download and install JDK 7.0 from this link.
- Download and install NetBeans 7.0.1 from this link.

Prerequisites

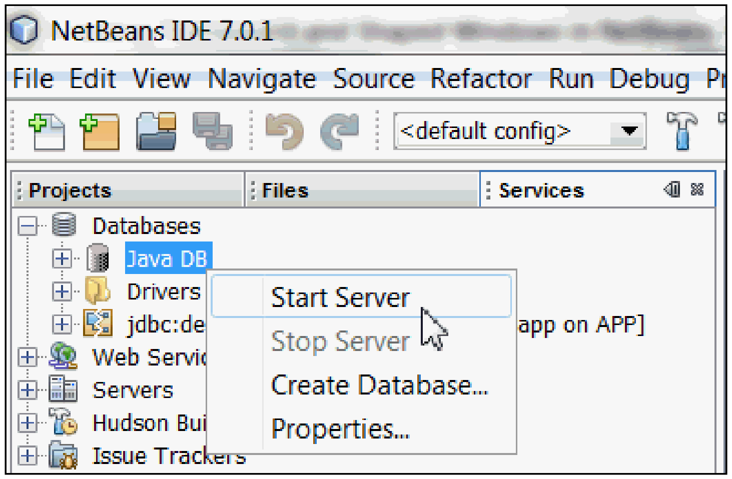
- Before starting this tutorial, you should have the software installed as listed under Software Requirements.
- NetBeans is running.
- Download and unzip the files.zip file that contains the file you need to perform this tutorial.

Creating a database connection

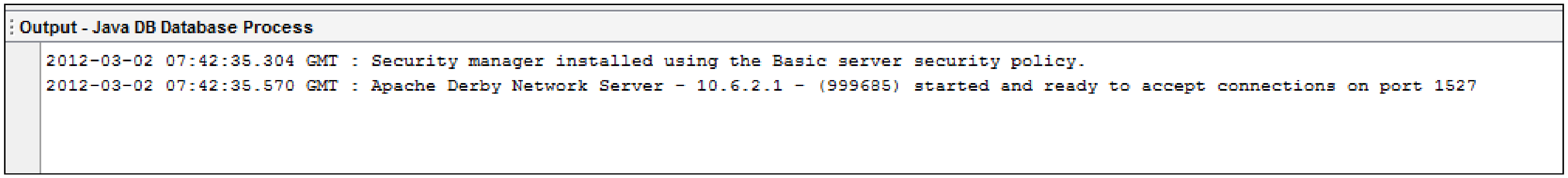
Java DB database server is part of NetBeans. W e will use Java DB as the database server . The following steps demonstrate creating the database **playerDB**.

1 . To start the Java DB Database from NetBeans, perform the following steps.

a. Click **Services** tab.
b. Expand **Databases** node.
c. Right-click **Java DB** icon.
d. Select **Start Server**.



Note the following output in the Output window , indicating that the DB server has started:

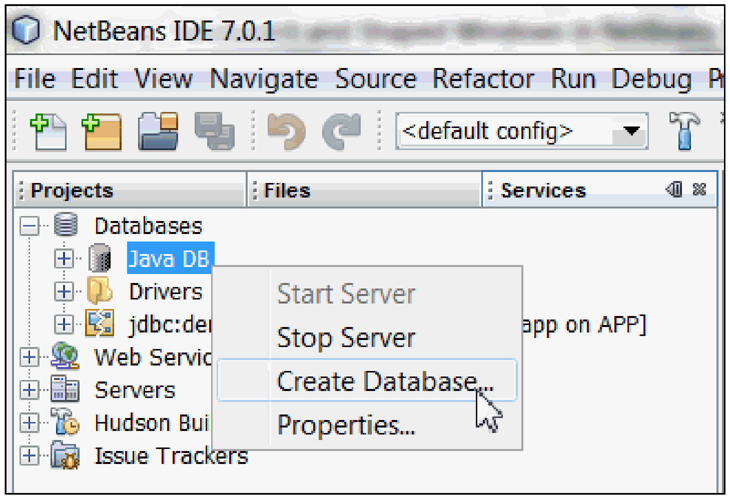

Output - Java DB Database Process

```
2012-03-02 07:42:35.304 GMT : Security manager installed using the Basic server security policy.
2012-03-02 07:42:35.570 GMT : Apache Derby Network Server - 10.6.2.1 - (999685) started and ready to accept connections on port 1527
```

Note that the DBserver version could vary from the version shown in the screenshot depending on the JDK build updates.

2. To create `playerDB` database, perform the following steps:

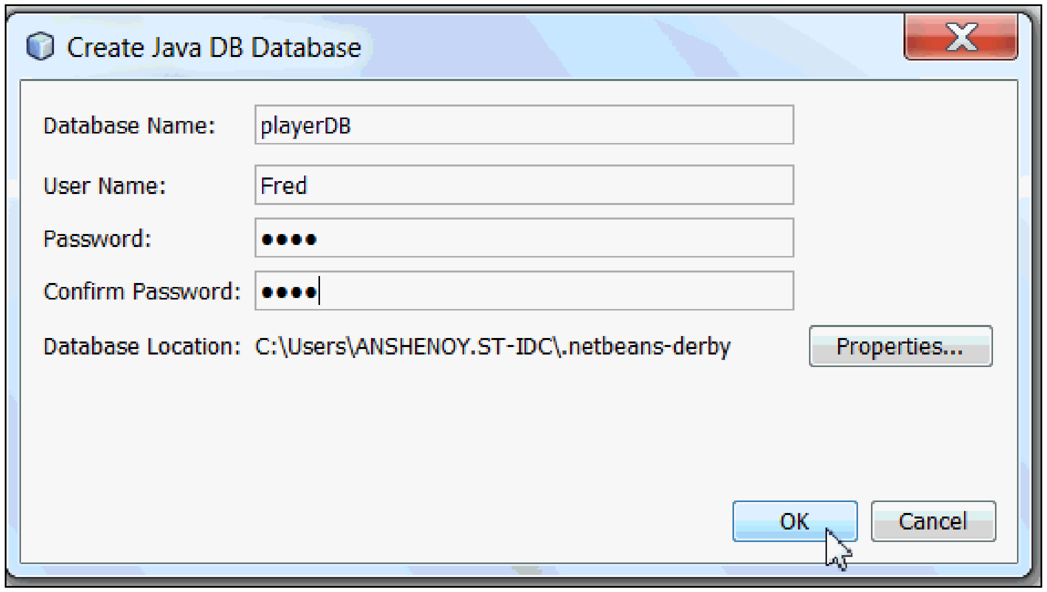
a. Right-click **Java DB** icon, select **Create Database**.



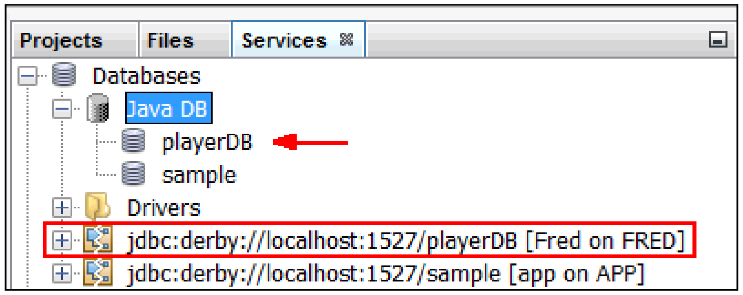
b. Enter the following information for the database:

Database Name: **playerDB**
User Name: **Fred**
Password: **Fred**
Confirm Password: **Fred**

c. Click OK.



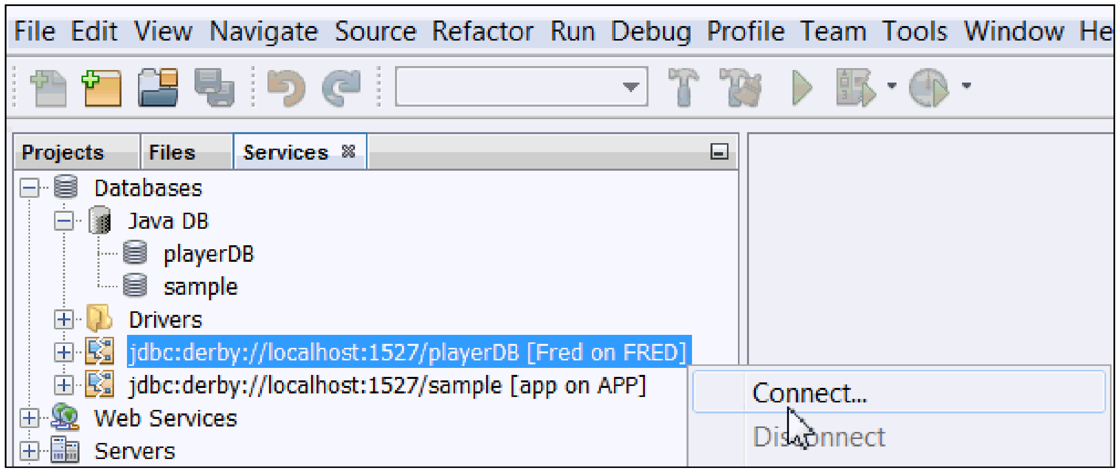
This creates the database and adds a connection for the database under the **Databases** icon.


Projects
Files
Services
Databases
Java DB
playerDB
sample
Drivers
jdbc:derby://localhost:1527/playerDB [Fred on FRED]
jdbc:derby://localhost:1527/sample [app on APP]

3 . To connect to the newly created database `playerDB`, perform the following steps :

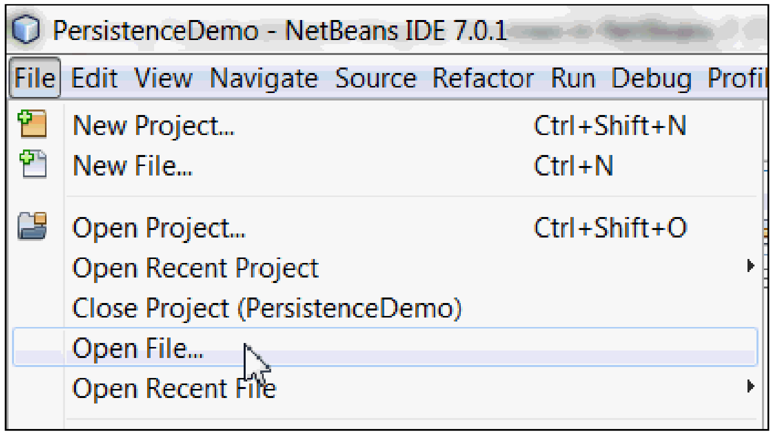
a. Right-click `jdbc:derby://localhost:1527/playerDB` connection.

b. Select **Connect**.


File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window He
Projects
Files
Services
Databases
Java DB
playerDB
sample
Drivers
jdbc:derby://localhost:1527/playerDB [Fred on FRED]
jdbc:derby://localhost:1527/sample [app on APP]
Web Services
Servers
Connect...
Disconnect

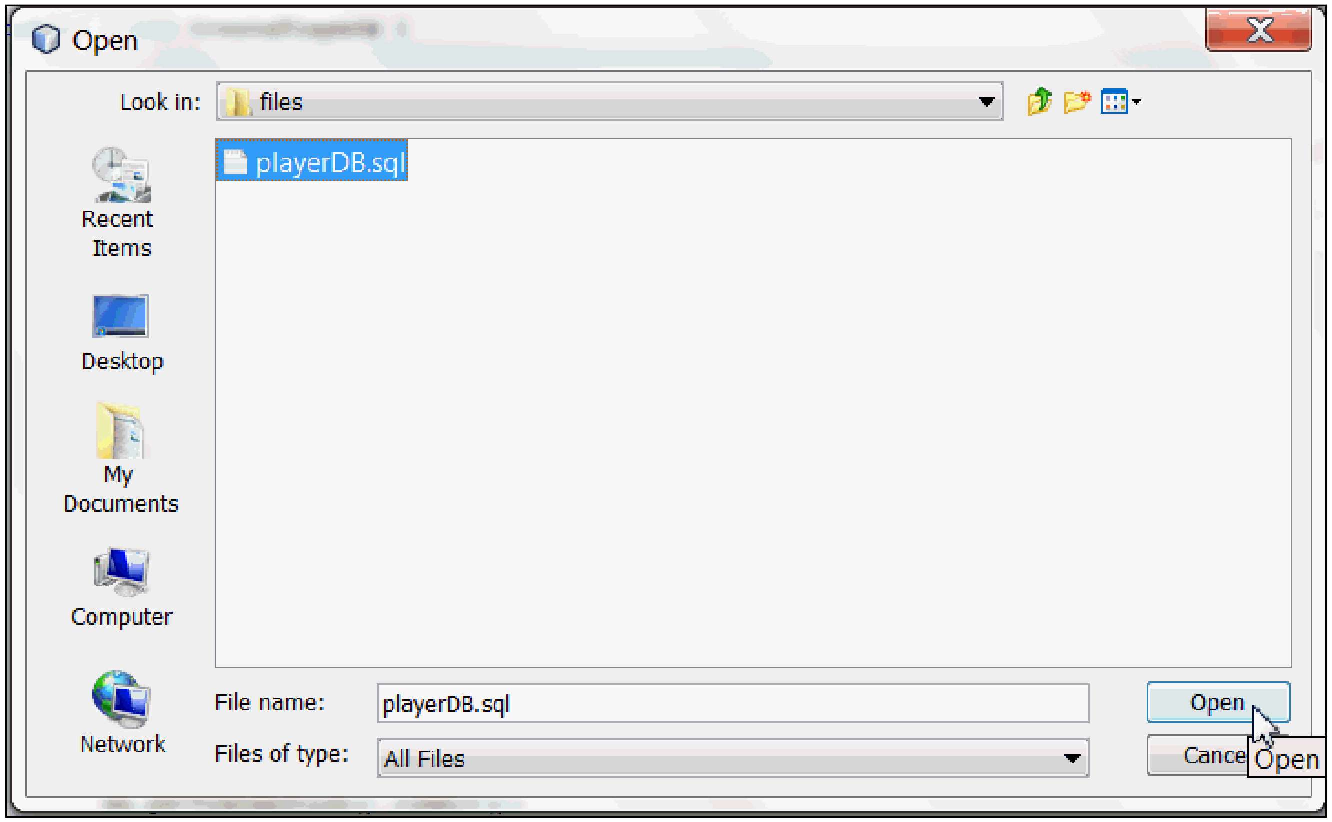
4. Create tables and populate them with data in `playerDB` database.

a. In NetBeans select File > Open File .


PersistenceDemo - NetBeans IDE 7.0.1
File Edit View Navigate Source Refactor Run Debug Profil
New Project... Ctrl+Shift+N
New File... Ctrl+N
Open Project... Ctrl+Shift+O
Open Recent Project
Close Project (PersistenceDemo)
Open File...
Open Recent File

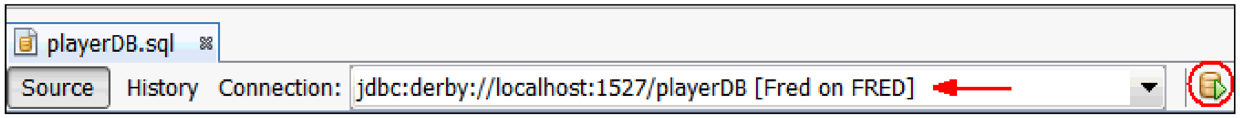
b. In the file browser navigate to the directory, where you unzipped the files from the Prerequisites section and select `playersDB.sql`

c. Click Open. The script automatically opens in the SQL Editor .



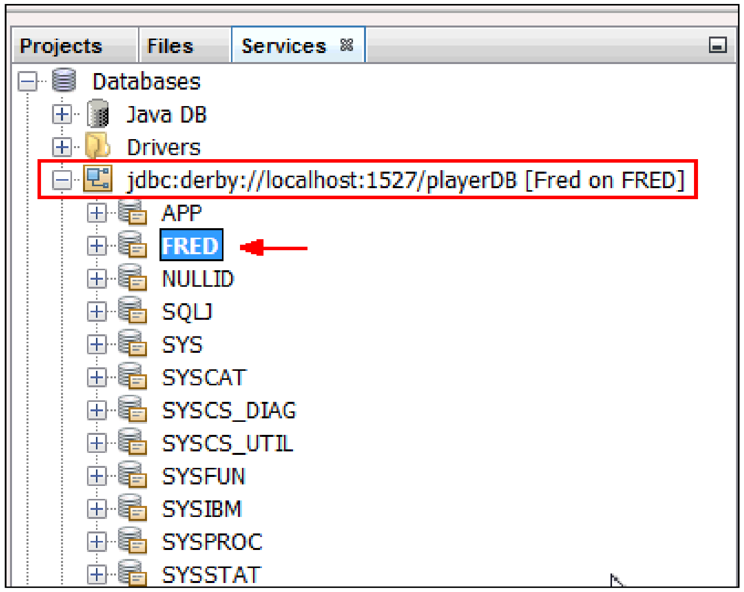
d. Select `jdbc:derby://localhost:1527/playerDB` in Connection drop-down box in the SQL Editor toolbar .

e. Click the **`Run SQL`** icon to execute the SQL statement.

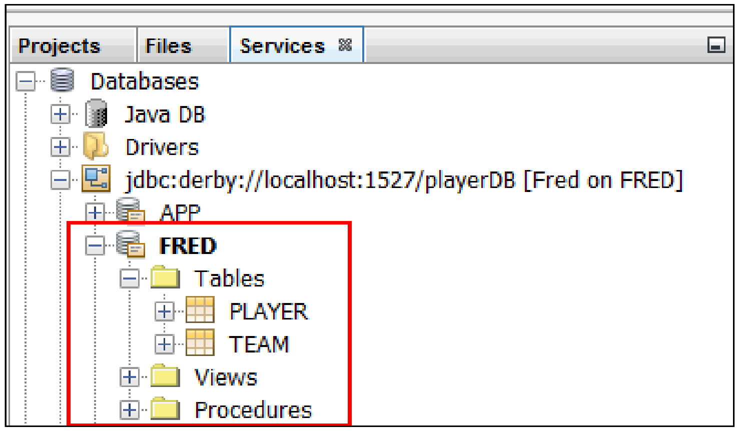

playerDB.sql
Source
History
Connection: jdbc:derby://localhost:1527/playerDB [Fred on FRED]

5. Examine the contents of the database.

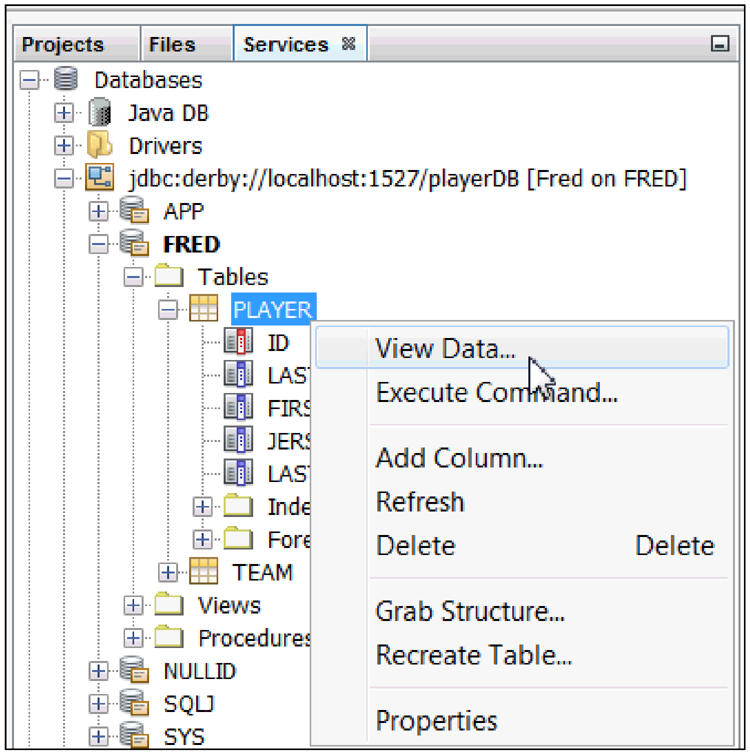
a. In the **Services** window, expand the `jdbc:derby://localhost:1527/playerDB`connection under the Databases node.
b. Right-click the connection and select **Refresh**.
c. Expand the **FRED**schema. You see the nodes for the Tables, Views, and Procedures.



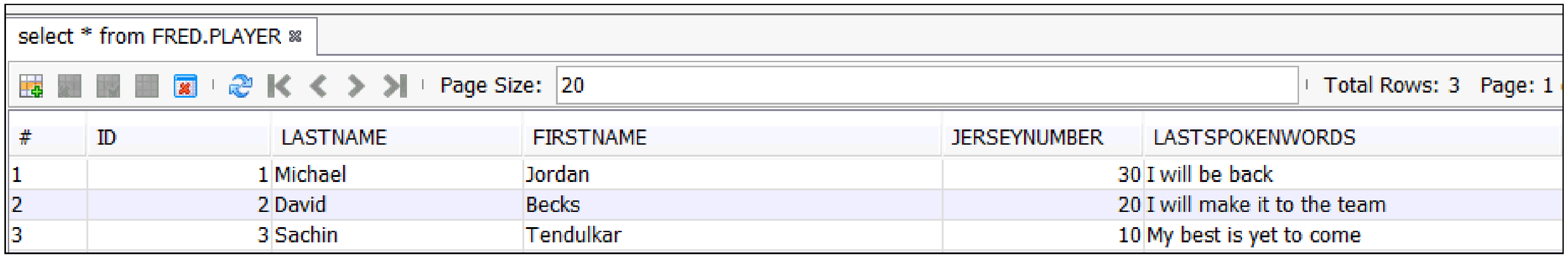
d. Expand the Tables node to see the PLAYER, TEAM tables.



e. Right-click PLAYER table node and select **View Data**.



f. A SQL command window opens and executes an SQL command to display the data in the table.


select * from FRED.PLAYER

Page Size: 20 Total Rows: 3 Page: 1

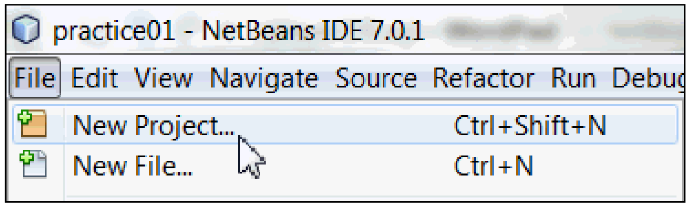
#	ID	LASTNAME	FIRSTNAME	JERSEYNUMBER	LASTSPOKENWORDS
1	1	Michael	Jordan	30	I will be back
2	2	David	Becks	20	I will make it to the team
3	3	Sachin	Tendulkar	10	My best is yet to come

g. Repeat the previous step for the TEAM table.

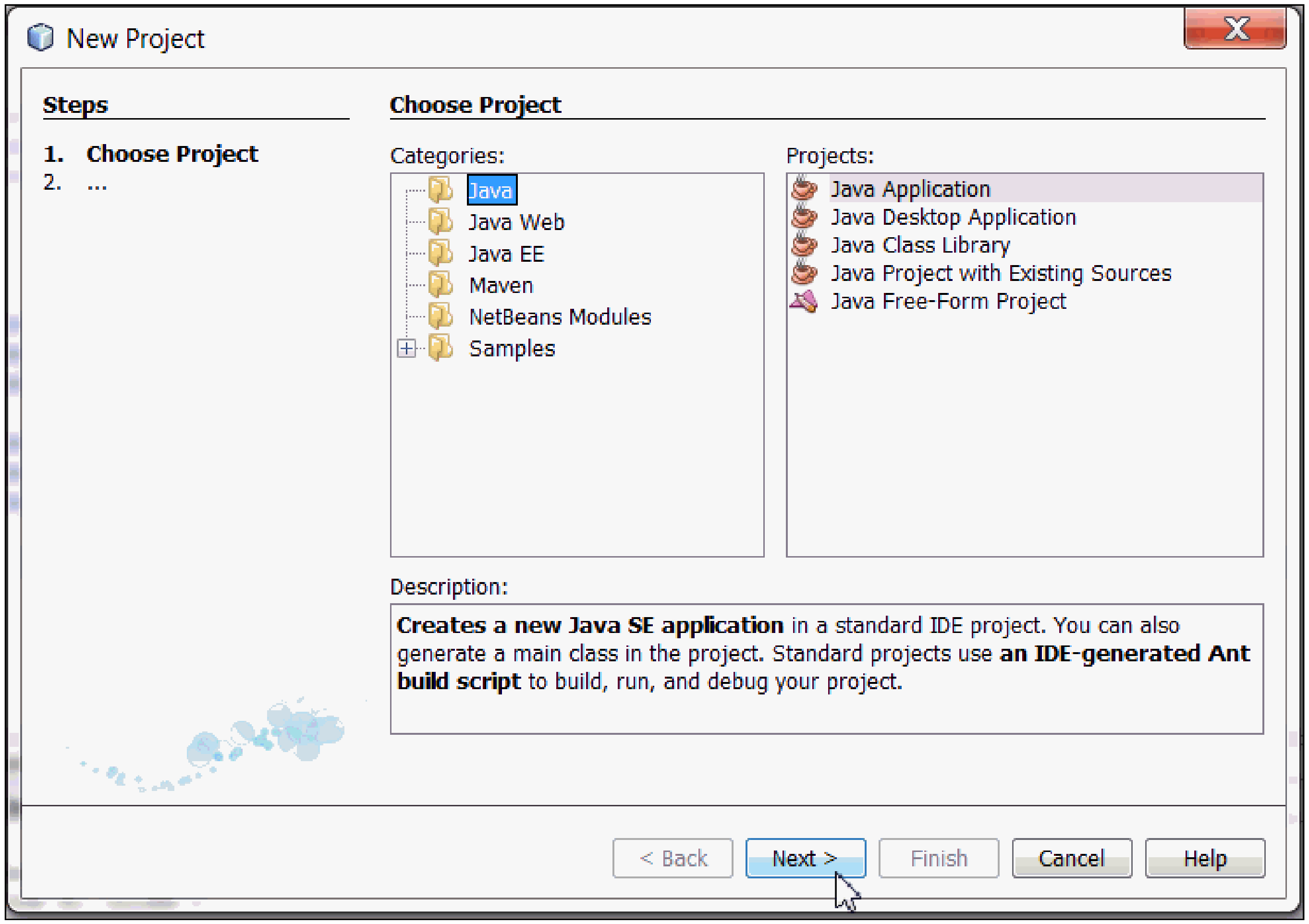
Generating Entity Classes from Database

The Java Persistence API requires that you identify the classes that you will store in a database. The API uses the term **entity** to define classes that it will map to a relational database. Y ou identify persistable entities and define their relationships using annotations. An entity represents a table in a relational database. Each entity instance corresponds to a row in the table. An Entity is coded as a POJO.

1. Create new Java Project .Select File > New Project .


practice01 - NetBeans IDE 7.0.1
File Edit View Navigate Source Refactor Run Debug
New Project... Ctrl+Shift+N
New File... Ctrl+N

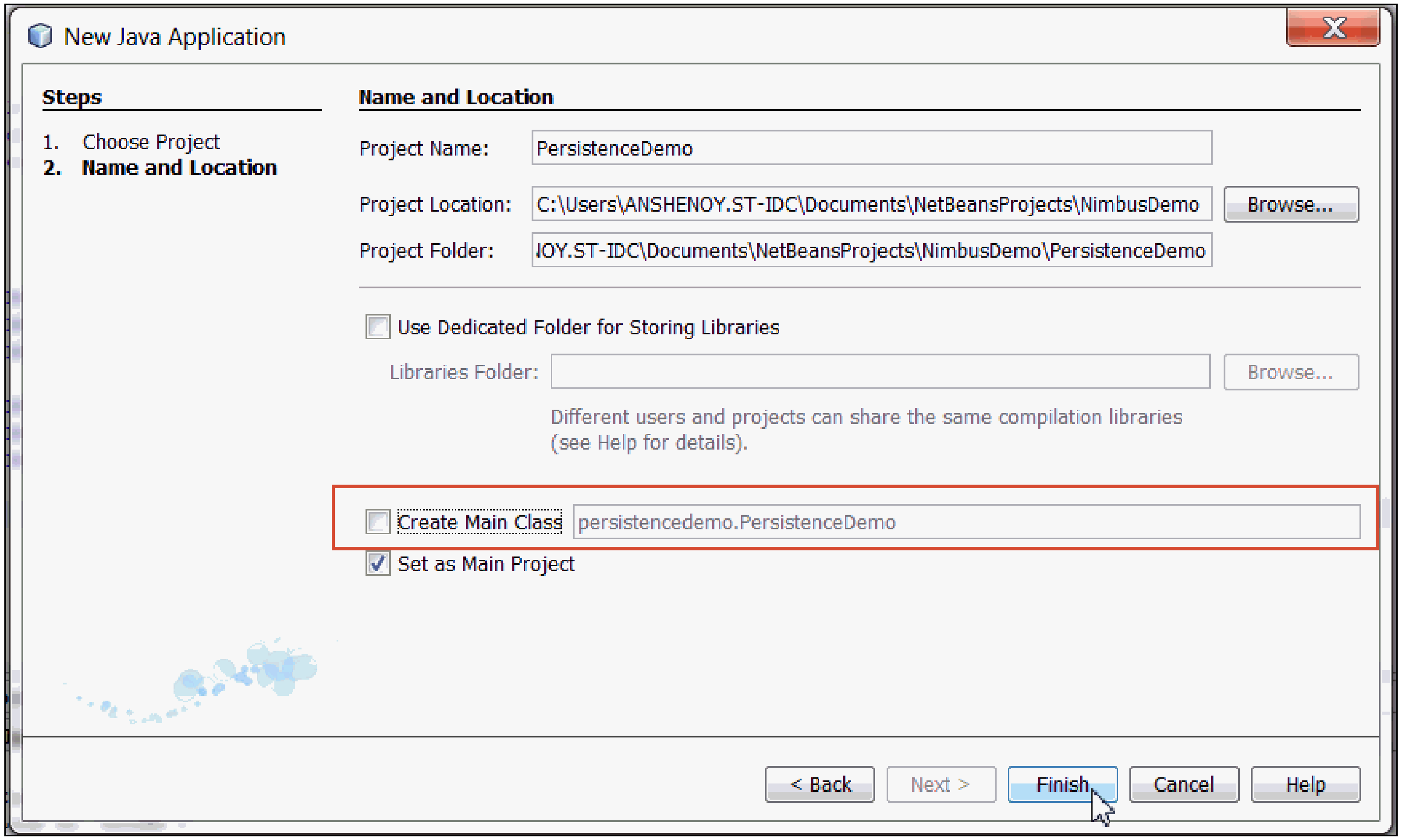
2. Select **Java** from the **Categories** column and **Java Application** from the **Projects** column and then click **Next**.



Creates a new Java SE application in a standard IDE project. You can also generate a main class in the project. Standard projects use **an IDE-generated Ant build script** to build, run, and debug your project.

3. Perform the following steps:

a. Name the project **PersistenceDemo**.
b. Uncheck the Create Main Class check box.
c. Click Finish



New Java Application

Steps
1. Choose Project
2. **Name and Location**

Name and Location

Project Name: PersistenceDemo

Project Location: C:\Users\ANSHENOY.ST-IDC\Documents\NetBeansProjects\NimbusDemo

Project Folder: IOY.ST-IDC\Documents\NetBeansProjects\NimbusDemo\PersistenceDemo

[ ] Use Dedicated Folder for Storing Libraries

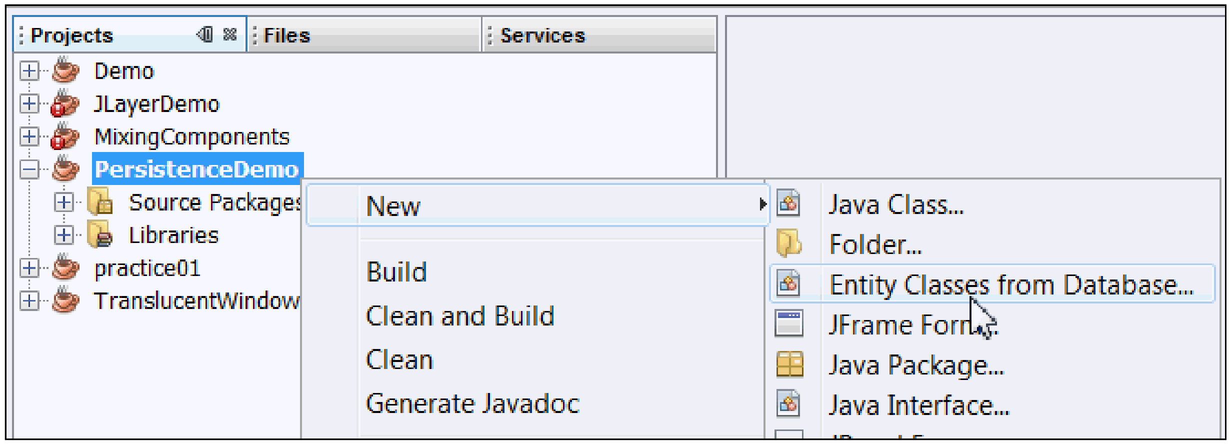
Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

[ ] Create Main Class persistencedemo.PersistenceDemo

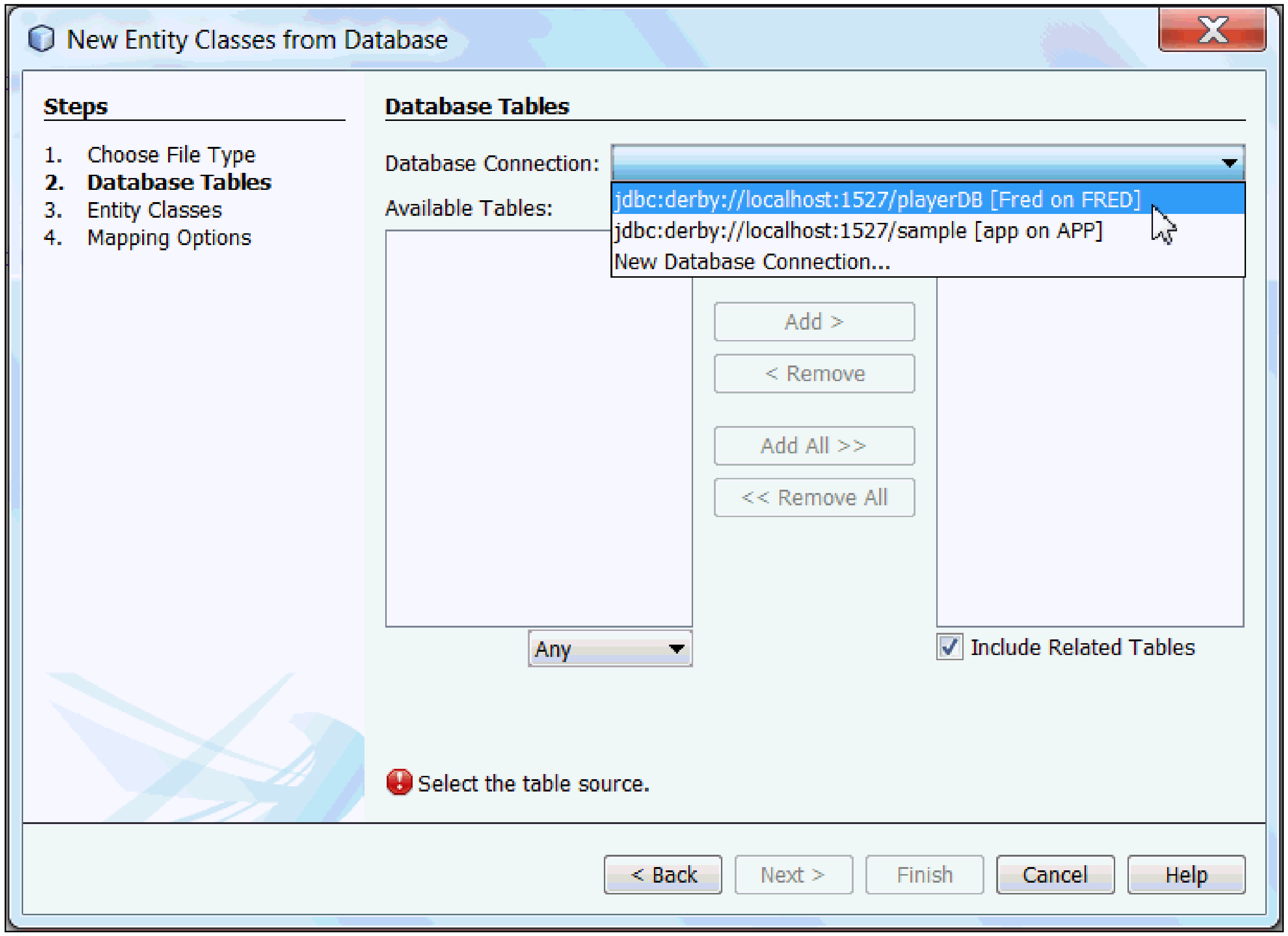
[✓] Set as Main Project

4. Right-click **PersistenceDemo** Project and select ***New > Entity Classes From Database***.



5. Enter the following information to create Entity classes:

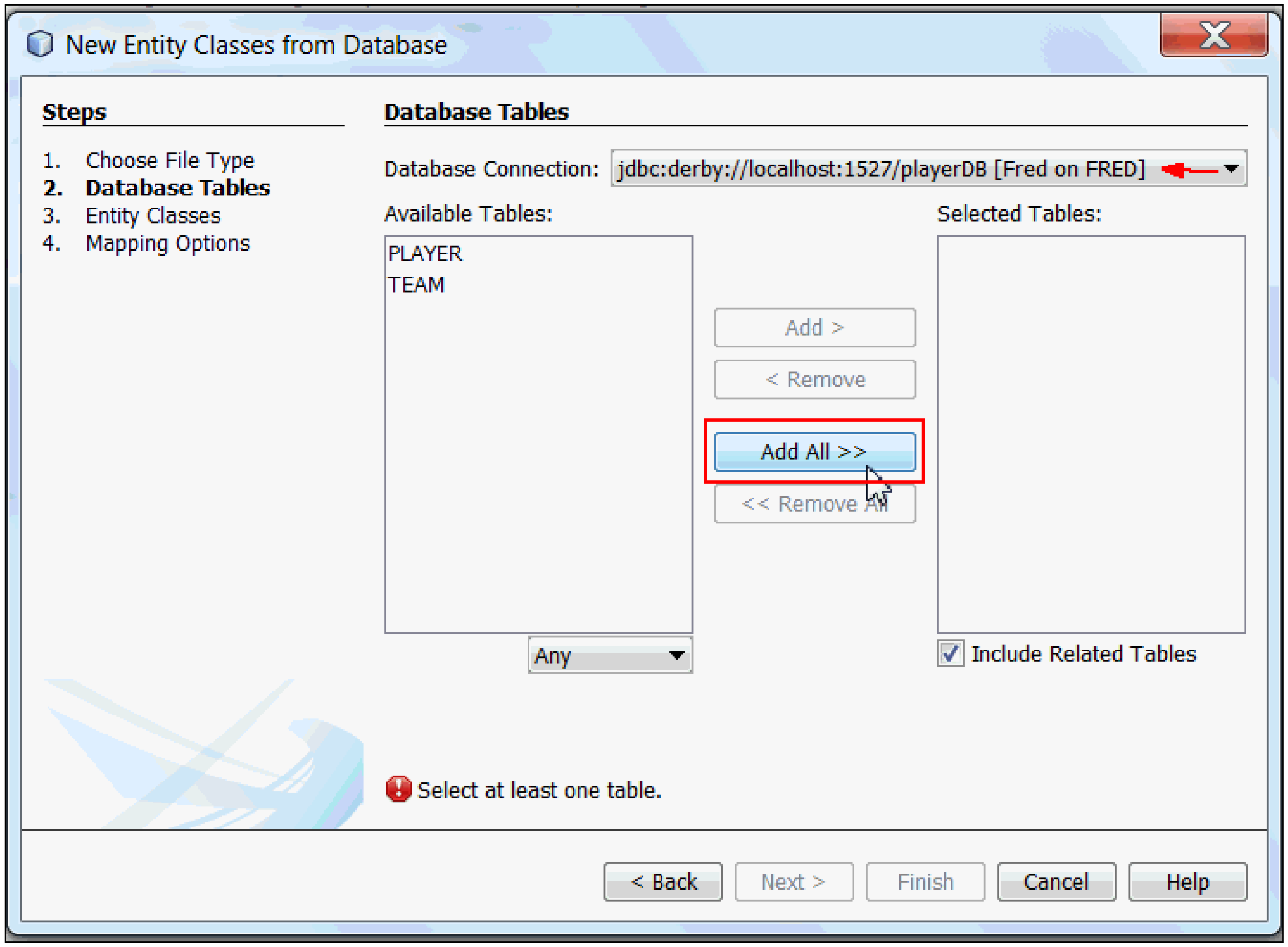
   a. In the **Database Connection** field select `jdbc:derby://localhost:1527/playerDB[Fredon FRED]` from the drop-down.



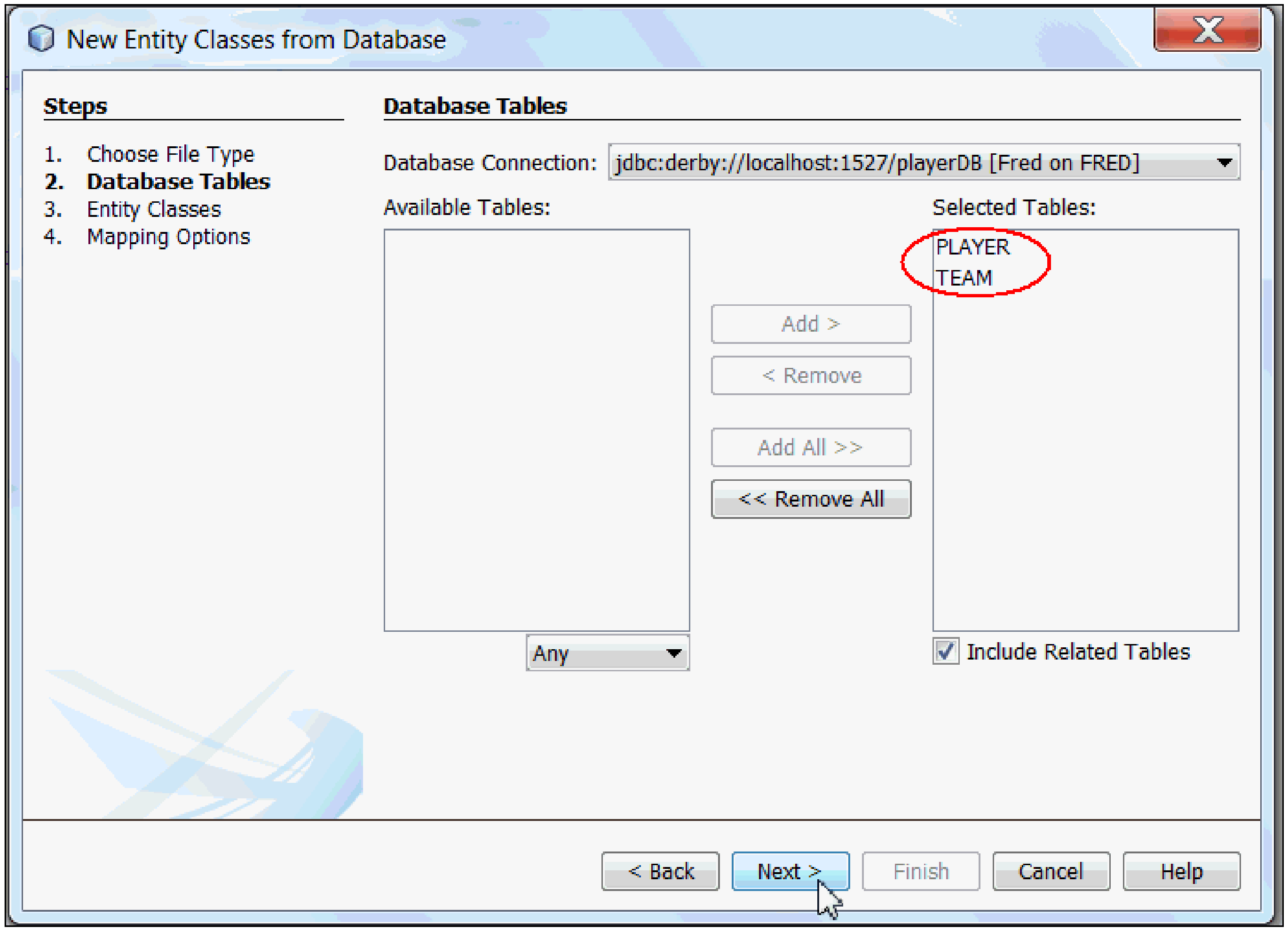
jdbc:derby://localhost:1527/playerDB [Fred on FRED]
jdbc:derby://localhost:1527/sample [app on APP]
New Database Connection...

Select the table source.

b. You see PLAYER and TEAM tables in Available Tables category
c. Click Add All



d. You see both the tables PLA YER and TEAM in Selected Tables Category
e. Click Next


New Entity Classes from Database

Steps
1. Choose File Type
2. **Database Tables**
3. Entity Classes
4. Mapping Options

Database Tables

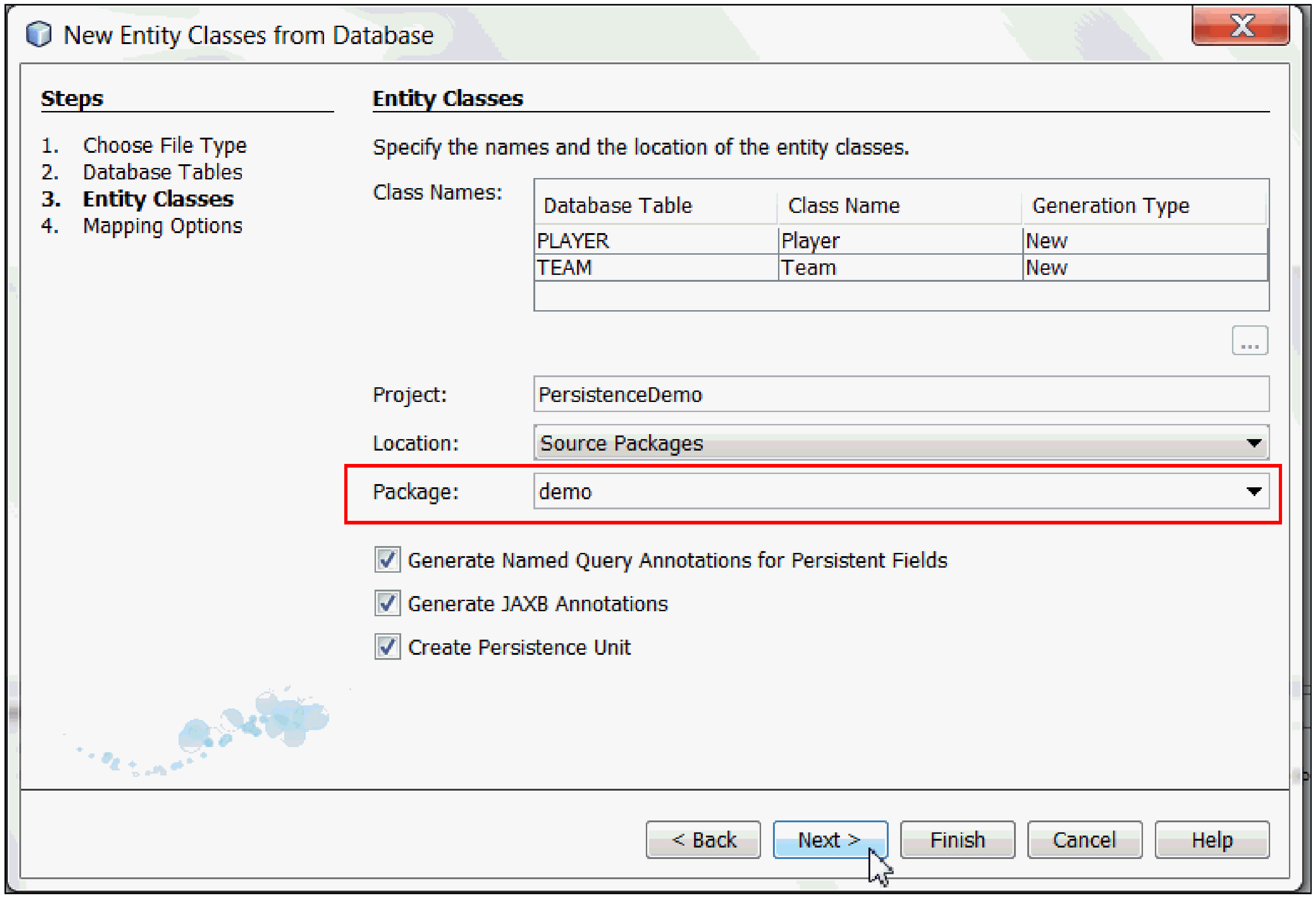
Database Connection: jdbc:derby://localhost:1527/playerDB [Fred on FRED]

Available Tables:

Selected Tables:
PLAYER
TEAM

[✓] Include Related Tables

6. In the Entity classes Window , enter the Package Name as **demo** and click Next.


New Entity Classes from Database

Steps

1. Choose File Type
2. Database Tables
3. **Entity Classes**
4. Mapping Options

Entity Classes

Specify the names and the location of the entity classes.

Class Names:

Database Table	Class Name	Generation Type
PLAYER	Player	New
TEAM	Team	New

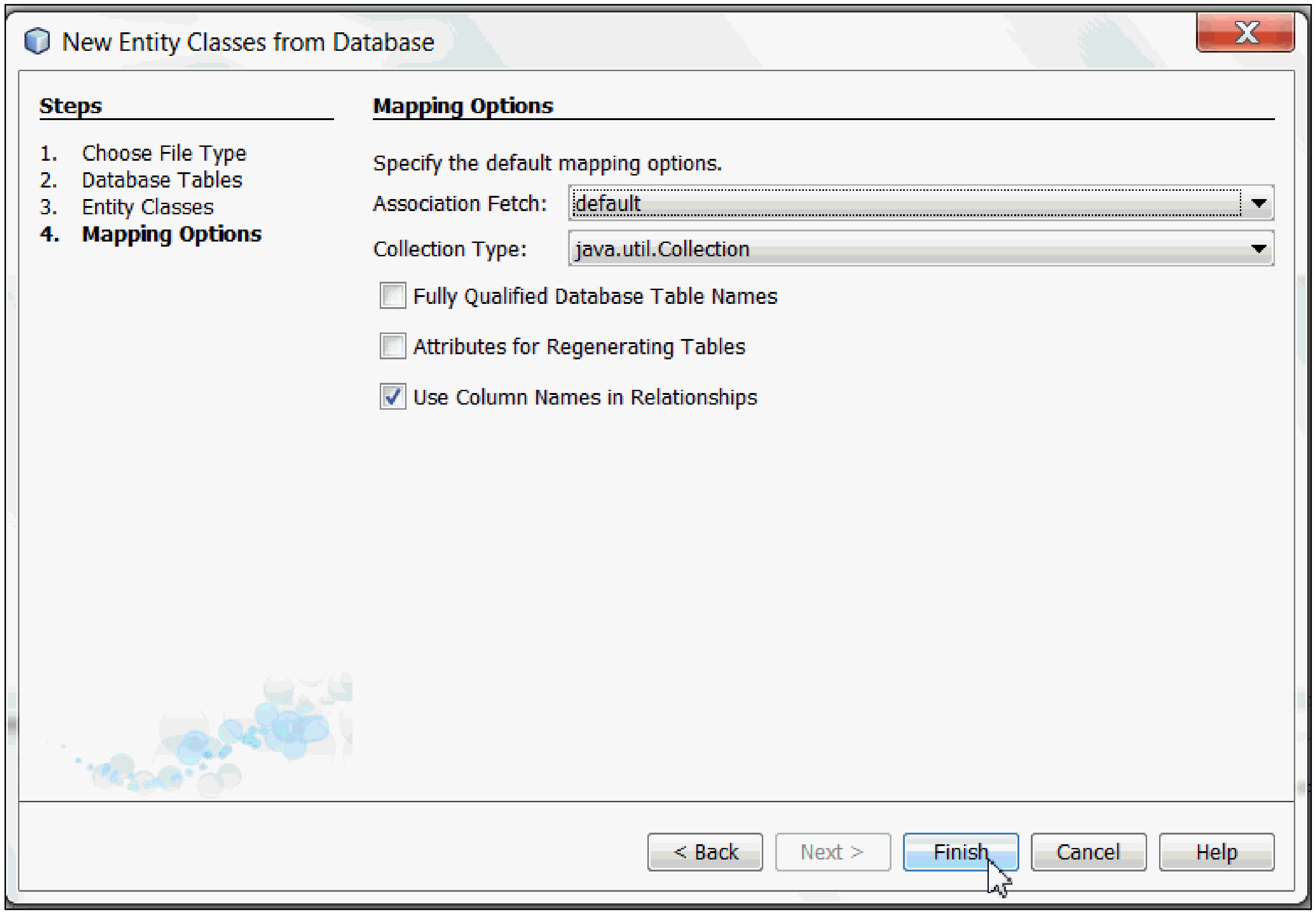
Project: PersistenceDemo

Location: Source Packages

Package: demo

- [✓] Generate Named Query Annotations for Persistent Fields
- [✓] Generate JAXB Annotations
- [✓] Create Persistence Unit

7. In the Mappings Window , click Finish with default selection.


New Entity Classes from Database

Steps

1. Choose File Type
2. Database Tables
3. Entity Classes
4. **Mapping Options**

Mapping Options

Specify the default mapping options.

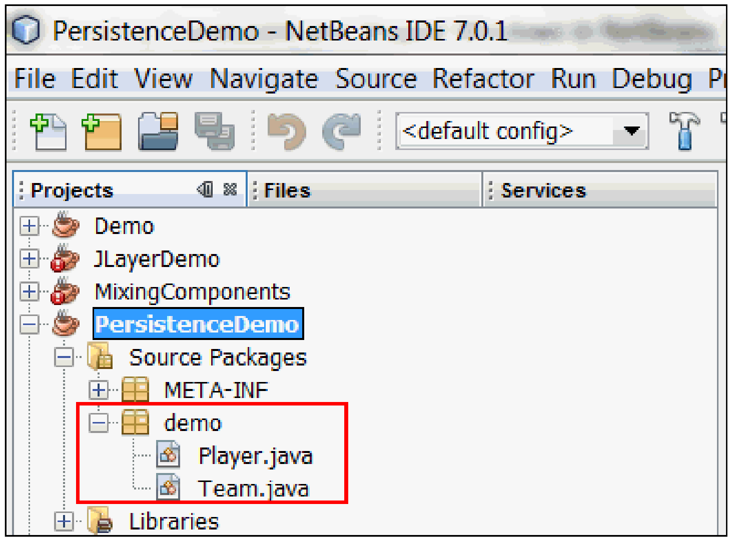
Association Fetch: default

Collection Type: java.util.Collection

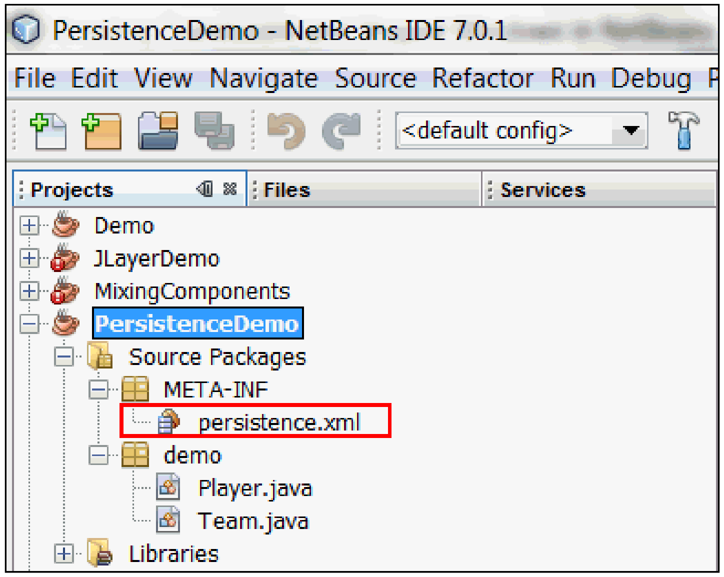
- [ ] Fully Qualified Database Table Names
- [ ] Attributes for Regenerating Tables
- [x] Use Column Names in Relationships

8. Verify the creation of Entity Classes.

a. Select the **PersistentDemo** Project.
b. Expand the demo package, you see `Team.java` and `Player.java` created.

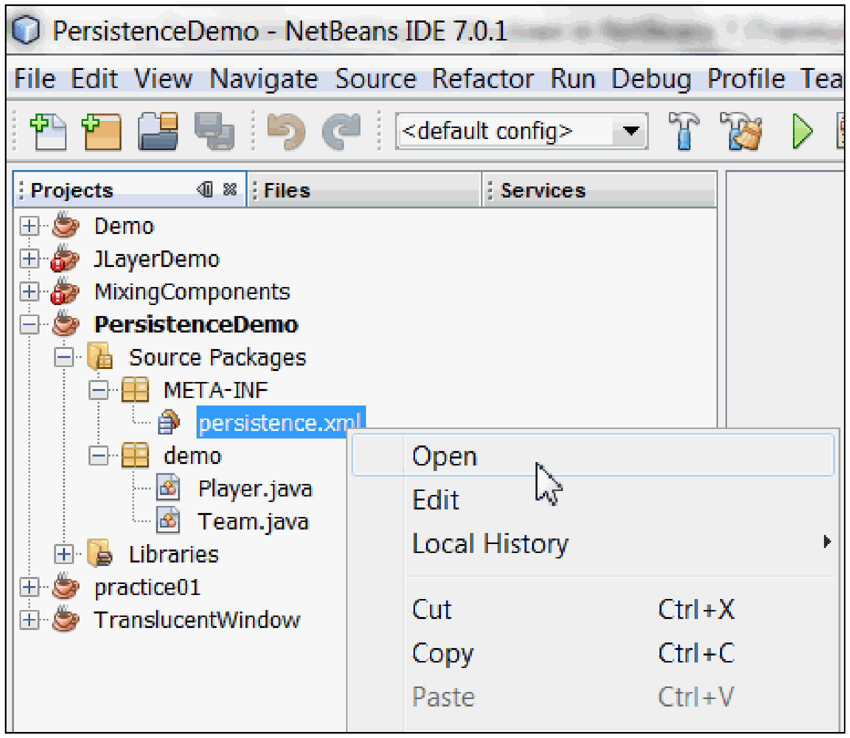


The above set of entities created in the application is called a **persistence unit**. Persistence units are configured in an XML file placed in the META-INF folder. It is used to specify the persistence provider name, entity class names and properties like the database connection URL, driver, user, password.

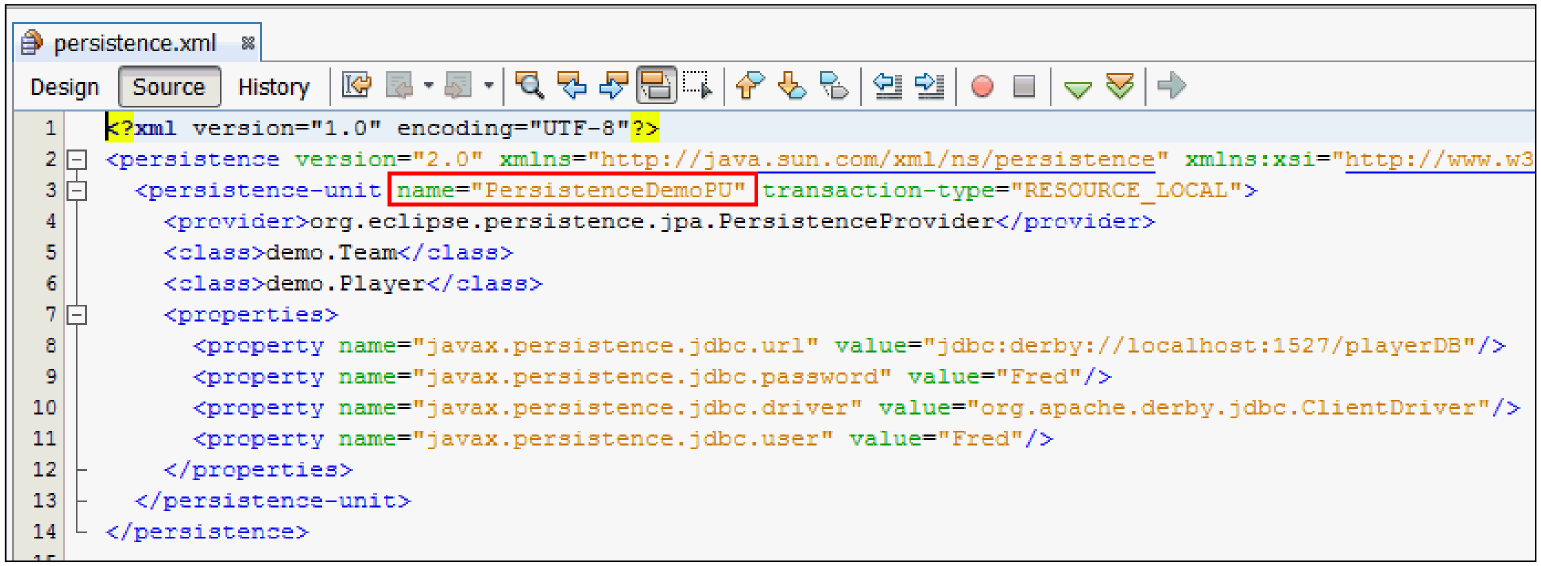


9. Name the **Persistence Unit** name as **PersistenceDemoPU** in the `persistence.xml` file.

a. Right-click `persistence.xml` and select **Open** to view it in the code editor window.



b. Select **Source** tab .
c. Verify the name of persistence Unit is `PersistenceDemoPU`as shown below .



```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3
  <persistence-unit name="PersistenceDemoPU" transaction-type="RESOURCE_LOCAL">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <class>demo.Team</class>
    <class>demo.Player</class>
    <properties>
      <property name="javax.persistence.jdbc.url" value="jdbc:derby://localhost:1527/playerDB"/>
      <property name="javax.persistence.jdbc.password" value="Fred"/>
      <property name="javax.persistence.jdbc.driver" value="org.apache.derby.jdbc.ClientDriver"/>
      <property name="javax.persistence.jdbc.user" value="Fred"/>
    </properties>
  </persistence-unit>
</persistence>
```

Implementing CRUD operations usingJPA

Entity objects are in-memory instances of entity classes which represent physical objects in the database. In JP A you can use entity objects for many operations, including Creating , Retrieving, Updating and Deleting database objects. We need three artifacts to implement the CRUD operations using JP A:

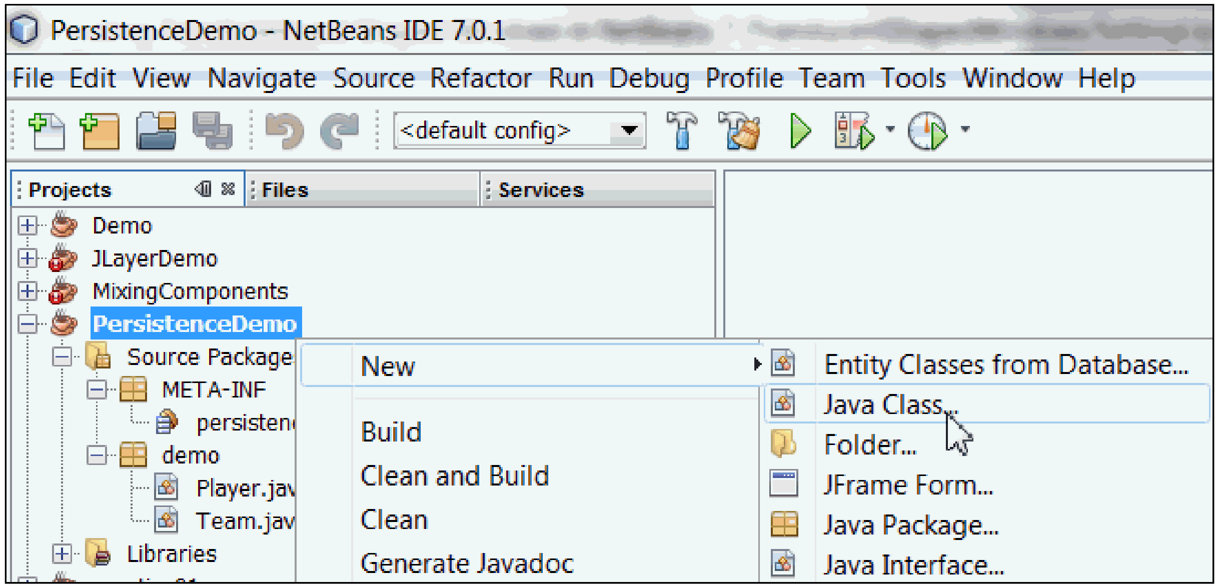
1. An Entity class
2. A persistence.xml file
3. A class (or client) through which we insert, update or find an entity .

The following section demonstrates Create operation, to persist Player entity objects into playerDB database. The Entity class, `Player.java` contains the mappings to the table, PLA YER in the form of annotations.

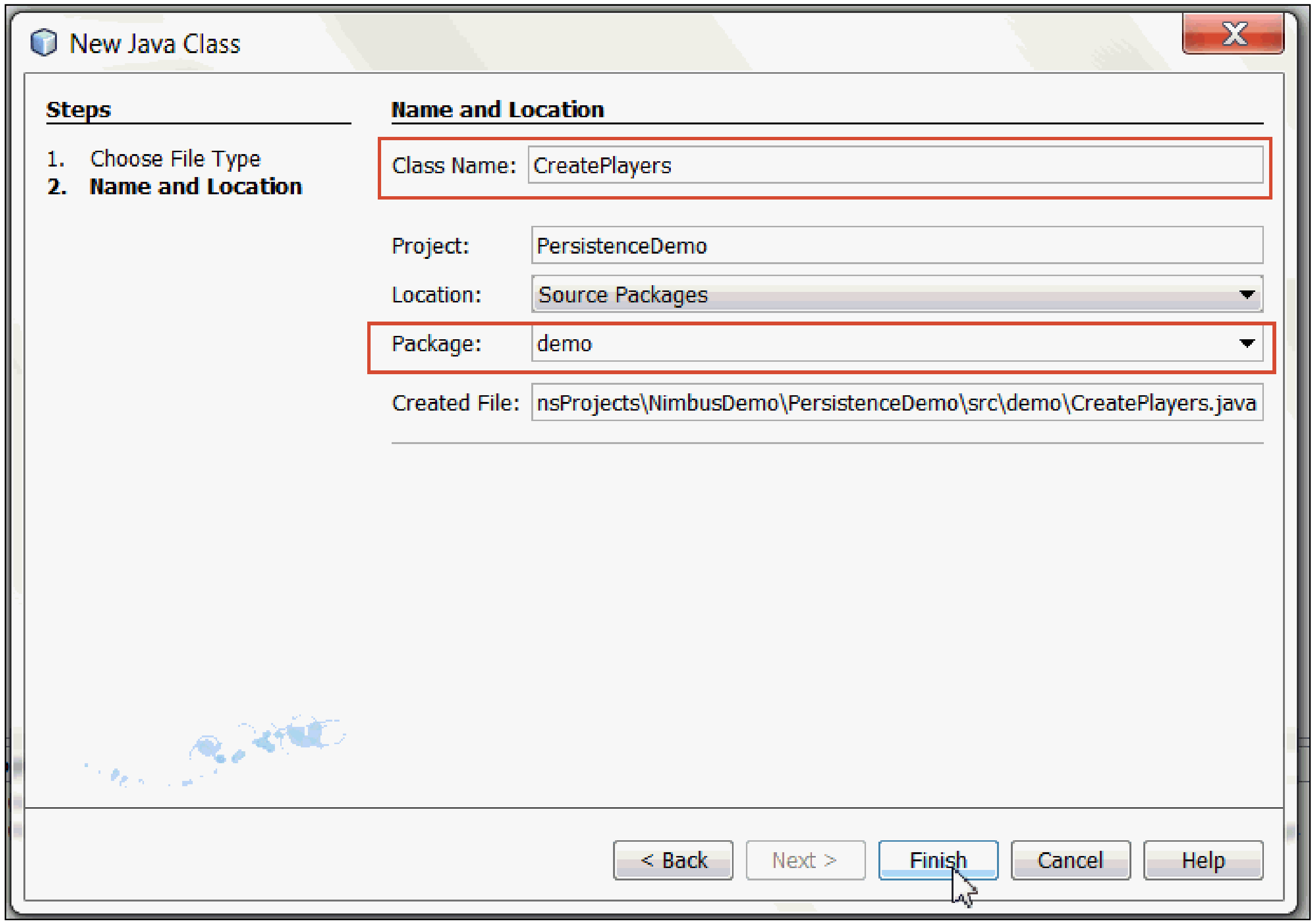
Create Operation

Create Operation

1 To create the client program `CreatePlayers.java` perform the following steps:
.
a. Right-click **PersistenceDemo** project and select New > Java Class.



b. Save the **class** as **CreatePlayers** and select **package name** as **demo**.



Class Name: CreatePlayers

Project: PersistenceDemo

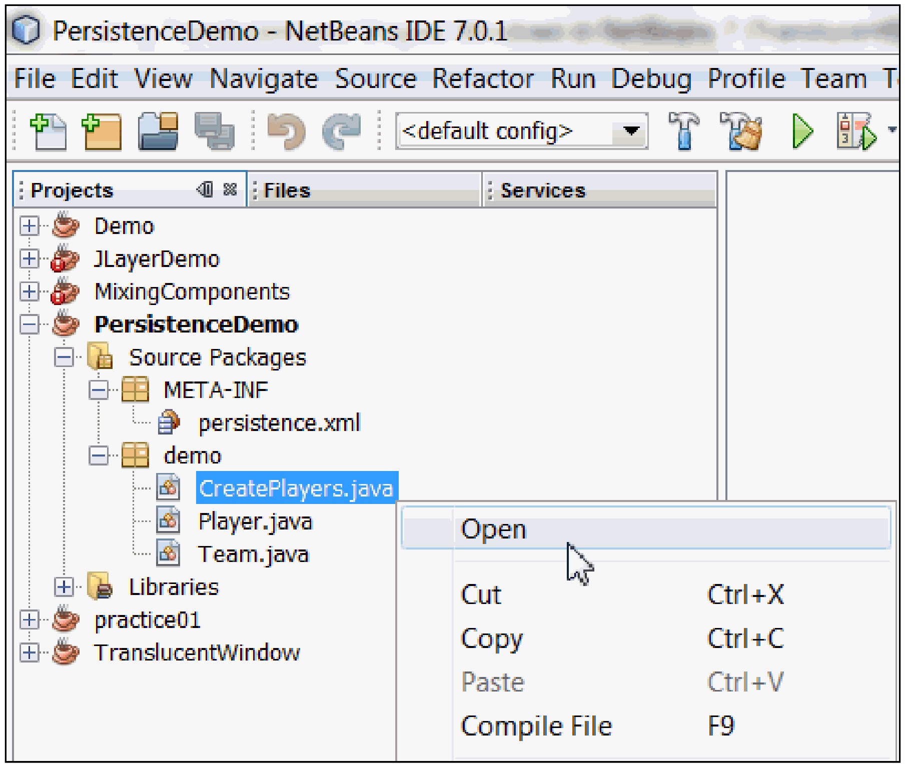
Location: Source Packages

Package: demo

Created File: nsProjects\NimbusDemo\PersistenceDemo\src\demo\CreatePlayers.java

c. Click Finish.

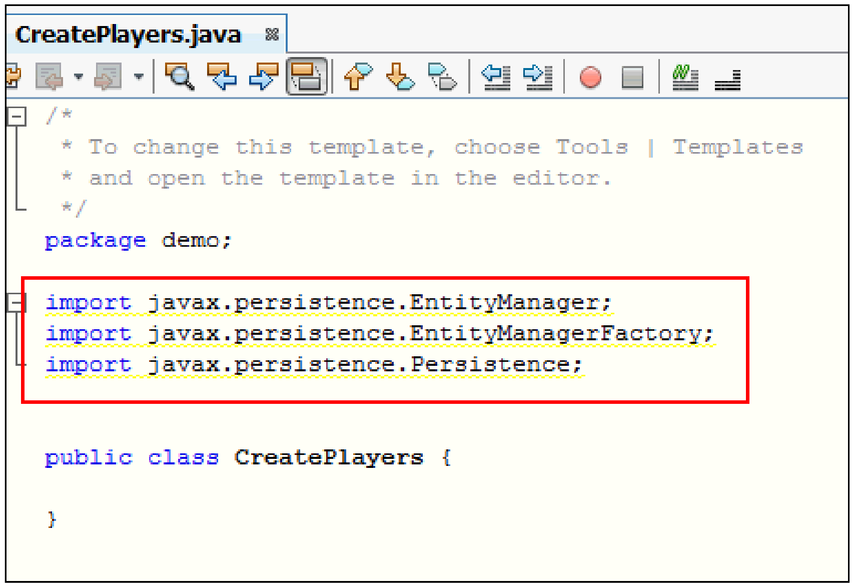
d. Right-click `CreatePlayers.java`and select **Open** to view it in the code editor window .



2 Import the following classes:

·

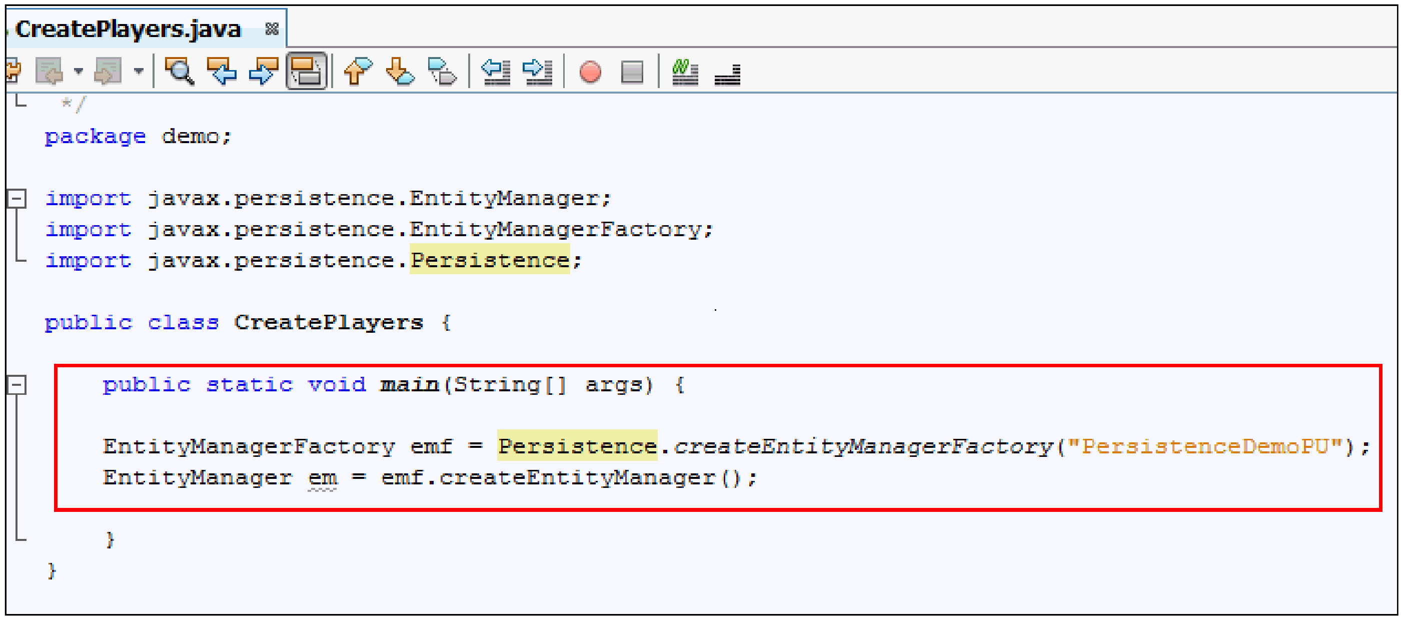
```
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
```



```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package demo;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;


public class CreatePlayers {

}
```

3. Create `main` method in the class and add the following lines of code.

```
public static void main(String[] args) {
EntityManagerFactory emf = Persistence.createEntityManagerFactory("PersistenceDemoPU");
EntityManager em = emf.createEntityManager();
}
```



CreatePlayers.java

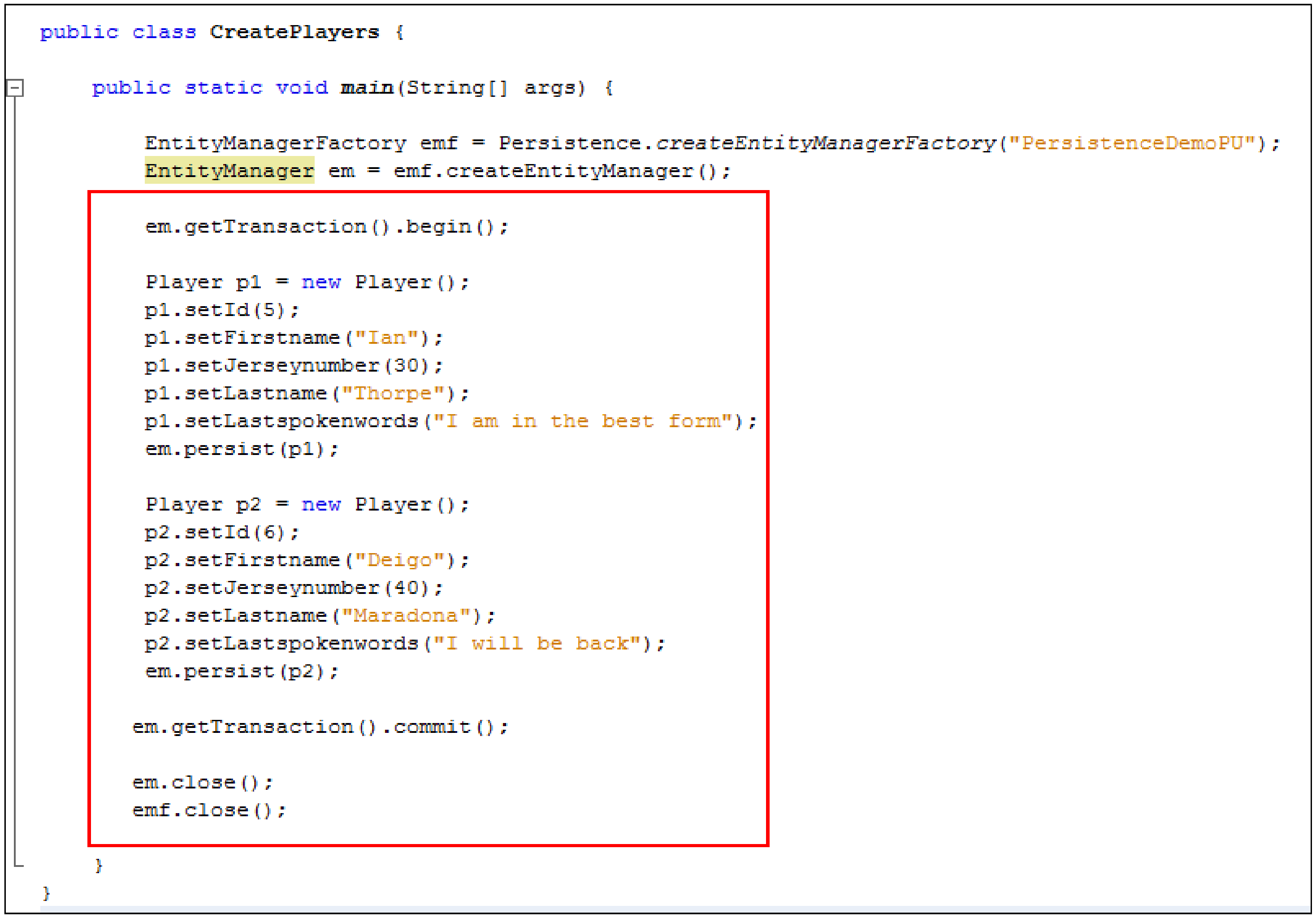
```
*/
package demo;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class CreatePlayers {

    public static void main(String[] args) {

    EntityManagerFactory emf = Persistence.createEntityManagerFactory("PersistenceDemoPU");
    EntityManager em = emf.createEntityManager();

    }
}
```

The above code demonstrates creation of an `EntityManager` instance. To persist a new entity, you need an `EntityManager` instance. `EntityManagerFactory` is a factory for creating an `EntityManager`. `EntityManager` manages entities and it is responsible for their addition, updating and deletion. Since EntityManager instances represent a persistence unit, you must provide the persistence unit name. In this example **`PersistenceDemoPU`** is the persistence unit name which is declared in the `persistence.xml` file along with other properties.

4. Add below code to the `main` method.

```
em.getTransaction().begin();

Player p1 = new Player();
p1.setId(5);
p1.setFirstname("Ian");
p1.setJerseynumber(30);
p1.setLastname("Thorpe");
p1.setLastspokenwords("I am in the best form");
em.persist(p1);

Player p2 = new Player();
p2.setId(6);
p2.setFirstname("Deigo");
p2.setJerseynumber(40);
p2.setLastname("Maradona");
p2.setLastspokenwords("I will be back");
em.persist(p2);

em.getTransaction().commit();

em.close();
emf.close();
```

The above code creates a transcation, 2 objects of the Player class which is persisted as 2 rows in the Player Table.

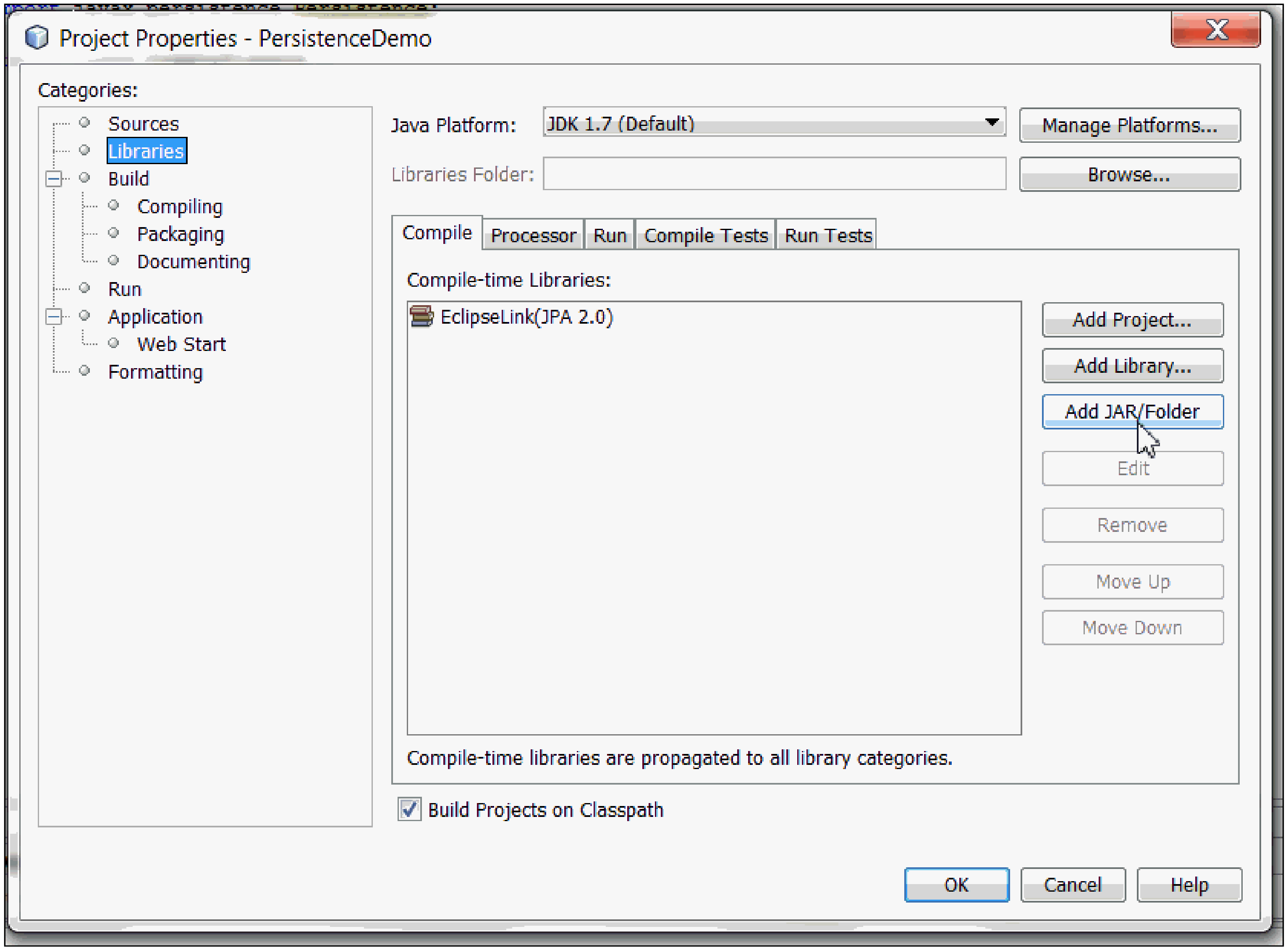


```
public class CreatePlayers {

    public static void main(String[] args) {

        EntityManagerFactory emf = Persistence.createEntityManagerFactory("PersistenceDemoPU");
        EntityManager em = emf.createEntityManager();

        em.getTransaction().begin();

        Player p1 = new Player();
        p1.setId(5);
        p1.setFirstname("Ian");
        p1.setJerseynumber(30);
        p1.setLastname("Thorpe");
        p1.setLastspokenwords("I am in the best form");
        em.persist(p1);

        Player p2 = new Player();
        p2.setId(6);
        p2.setFirstname("Deigo");
        p2.setJerseynumber(40);
        p2.setLastname("Maradona");
        p2.setLastspokenwords("I will be back");
        em.persist(p2);

       em.getTransaction().commit();

       em.close();
       emf.close();

    }
}
```

5 Add the **Java DB client jar** to connect to the Java DB database server. Complete the following steps:

1) Right-click Project > Project Properties> Libraries.

2) Select **Add JAR/Folder**



Java Platform: JDK 1.7 (Default)

Libraries Folder:

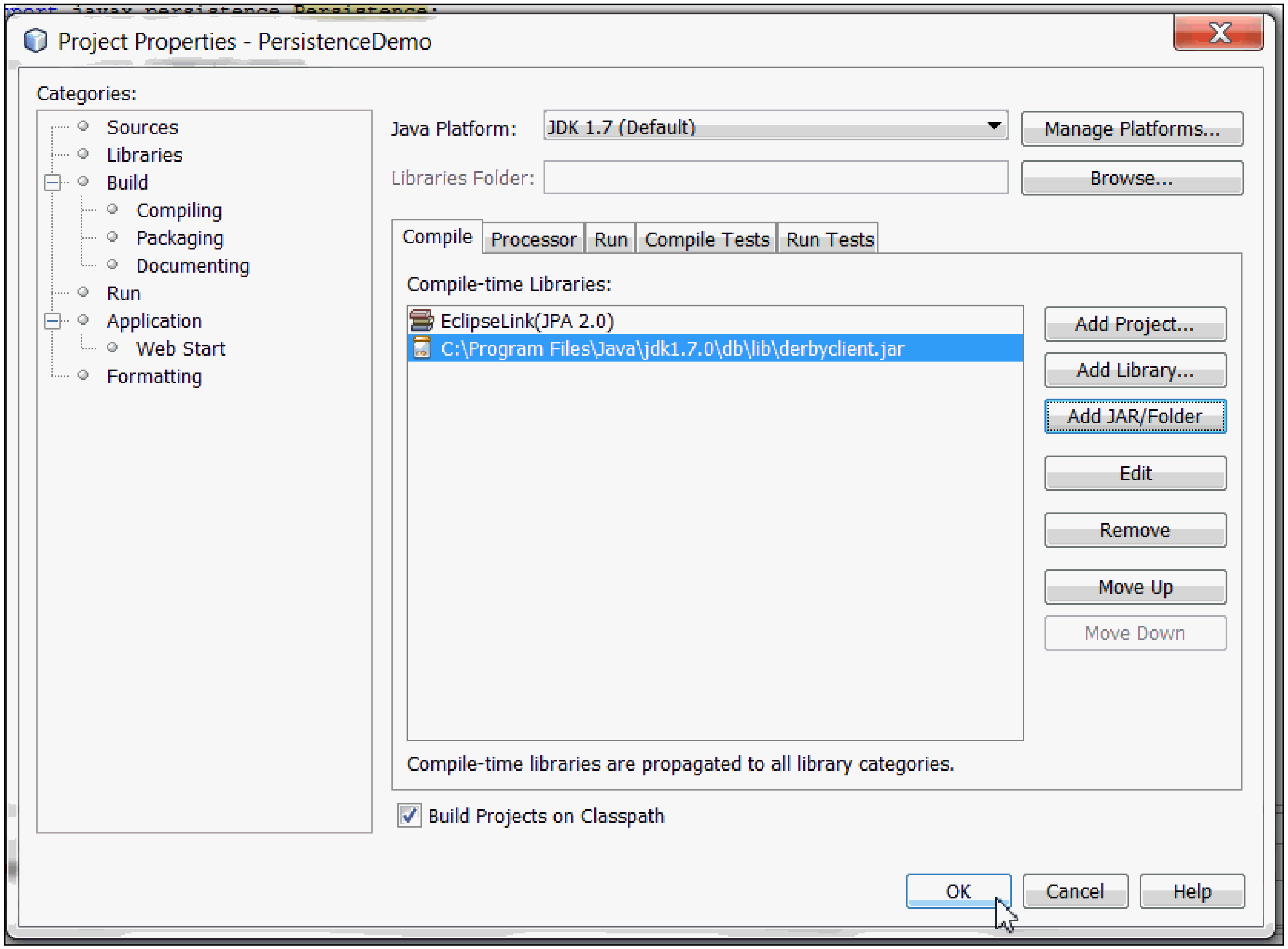
Compile-time Libraries:

EclipseLink(JPA 2.0)

Compile-time libraries are propagated to all library categories.

[✓] Build Projects on Classpath

3) Browse to `C:\Program Files\Java\jdk1.7.0_01\db\lib\derbyclient.jar`
4) Click Open > Click OK.



Java Platform: JDK 1.7 (Default)

Libraries Folder:

Compile-time Libraries:

EclipseLink(JPA 2.0)

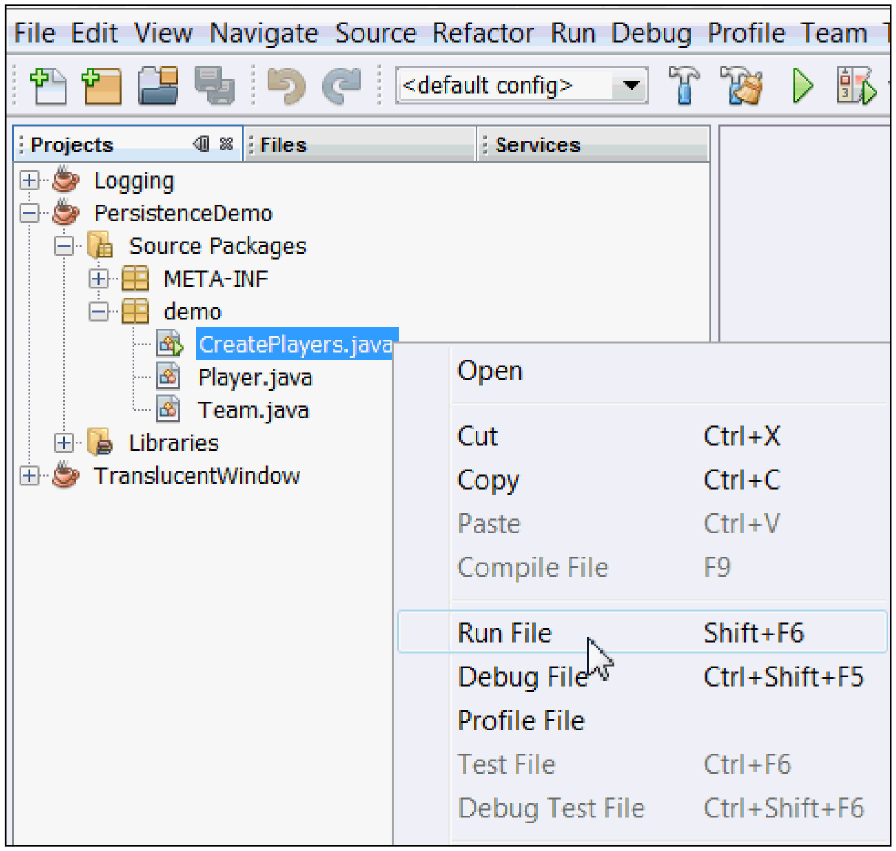
C:\Program Files\Java\jdk1.7.0\db\lib\derbyclient.jar

Compile-time libraries are propagated to all library categories.

☑ Build Projects on Classpath

6 In the Projects window , right-click `CreatePlayers.java`and select **Run File**from the right-click menu.

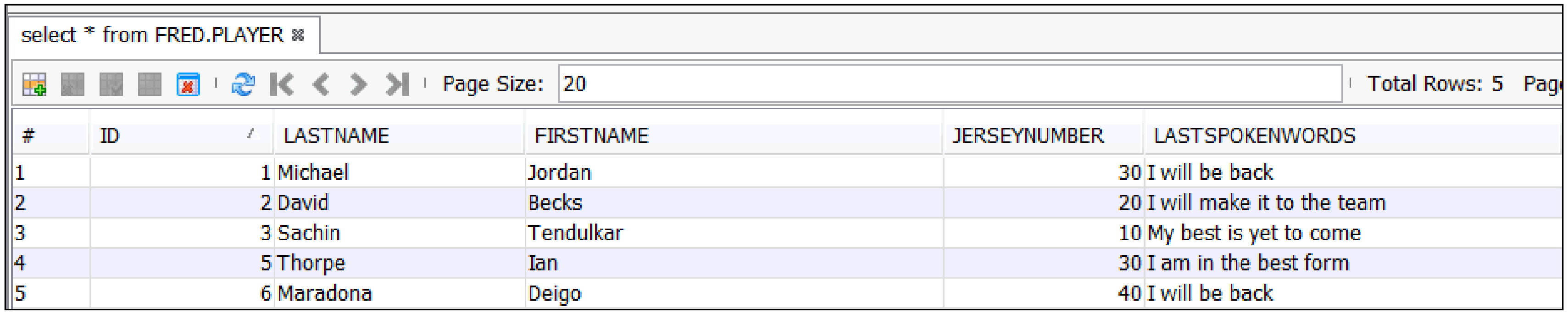
.



7 Verify the output. Examine the contents of the database.

a. In the **Services** window, expand the `jdbc:derby://localhost:1527/playerDB` connection under the Databases node.
b. Right-click the connection and select Refresh.
c. Expand the **FRED** schema > Expand Tables Node >PLAYER Table.
d. Right-click PLAYER table node and select **View Data**.

You see the the 2 rows inserted in the PLAYER table.



#	ID	LASTNAME	FIRSTNAME	JERSEYNUMBER	LASTSPOKENWORDS
1	1	Michael	Jordan	30	I will be back
2	2	David	Becks	20	I will make it to the team
3	3	Sachin	Tendulkar	10	My best is yet to come
4	5	Thorpe	Ian	30	I am in the best form
5	6	Maradona	Deigo	40	I will be back

Summary

This tutorial covers some of the capabilities of JP A in Java SE environment . The API simplifies object persistence by enabling use of POJOs throughout your application and in your database.

Resources

[Enterprise JavaBeans 3.0 specification](#)
[Java EE 6: Tutorial](#)

Credits

Curriculum Developer: Anjana Shenoy