Using RESTful Web Services to Implement CRUD Operations in NetBeans 7: Part 2

Purpose

This tutorial demonstrates using RESTful Web Services to implement CRUD database operations in NetBeans 7.

Time to Complete

Approximately 30 minutes.

Overview

RESTful Web Services use HTTP protocol methods. The HTTP methods such as GET, POST and DELETE can be mapped to create, read, update and delete (CRUD) actions to be performed.Java specification, JAX-RS (JSR 311) provides an API for creating RESTful web services in Java. Jersey is its official reference implementation. It provides additional features through its own APIs such as the Jersey Client API.

This tutorial is the second part of the tutorial, Part-1 of the tutorial**, Creating RESTful Web Services in NetBeans 7 : Part 1** covered the Creation and Testing of RESTful Web Services in NetBeans 7.

In this tutorial, you will

Create a Java client application to consume RESTful Web Services.
Implement CRUD operations on the Database using RESTful Web Services.

Prerequisites

Before starting this tutorial, you should have completed **Part 1** of this tutorial and met all its prerequisites.

Building a Client Application to consume RESTful Web Services

The following section demonstrates how to create a Java client application PlayerClient, to consume the RESTful webservices created in PlayerServer application.

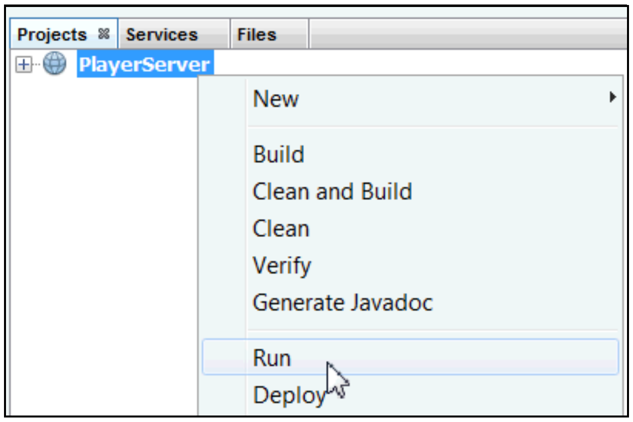
1 . Verify PlayerServer application is running as it creates the RESTful Web Services. Otherwise to start the Server, perform the following steps :

a. Open the Projects tab.

b. Right-click **PlayerServer** project.

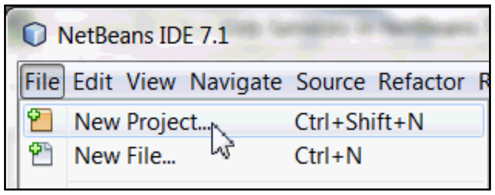
c. Select Run.

This action starts the GlassFish server and deploys the application.

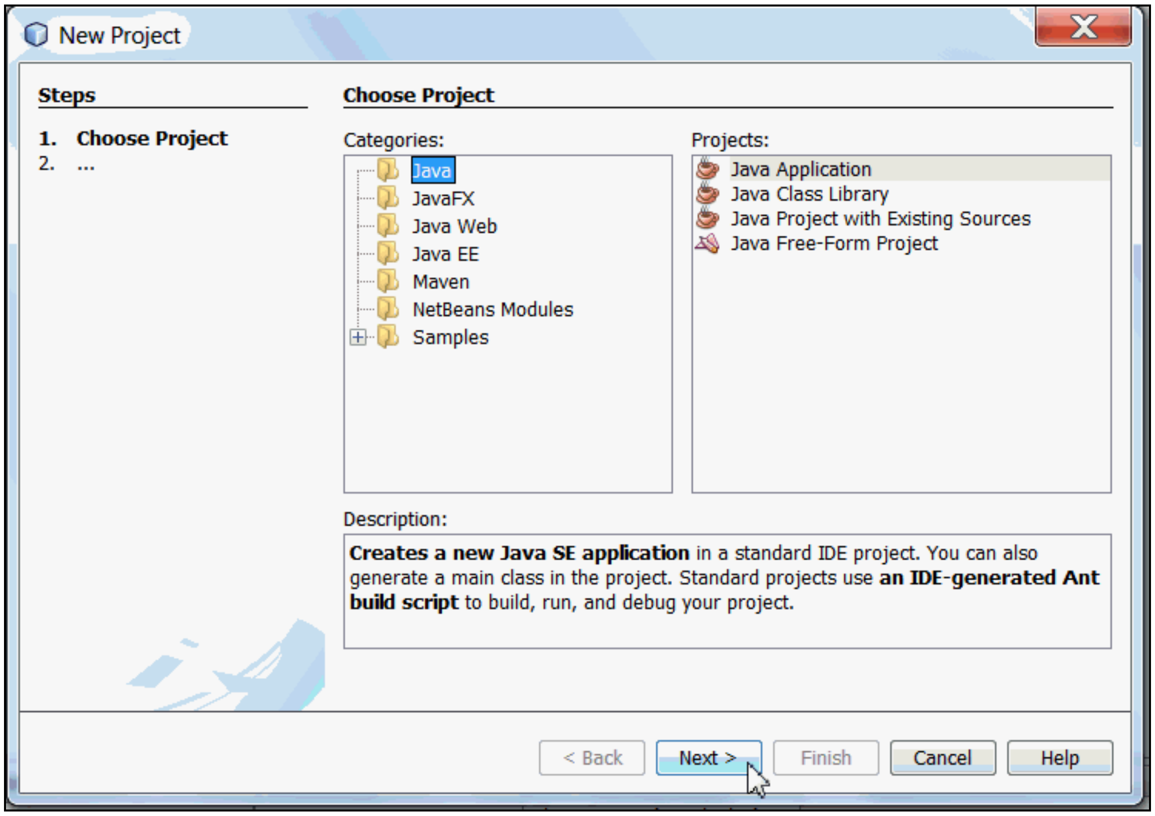


On successful deployment of the application - a default jsp page with url, `http://localhost:8080/PlayerServer/` is opened in the browser displaying "Hello Worl

2 . To create new Java Project, select File > New Project.


NetBeans IDE 7.1
File Edit View Navigate Source Refactor R
New Project... Ctrl+Shift+N
New File... Ctrl+N

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



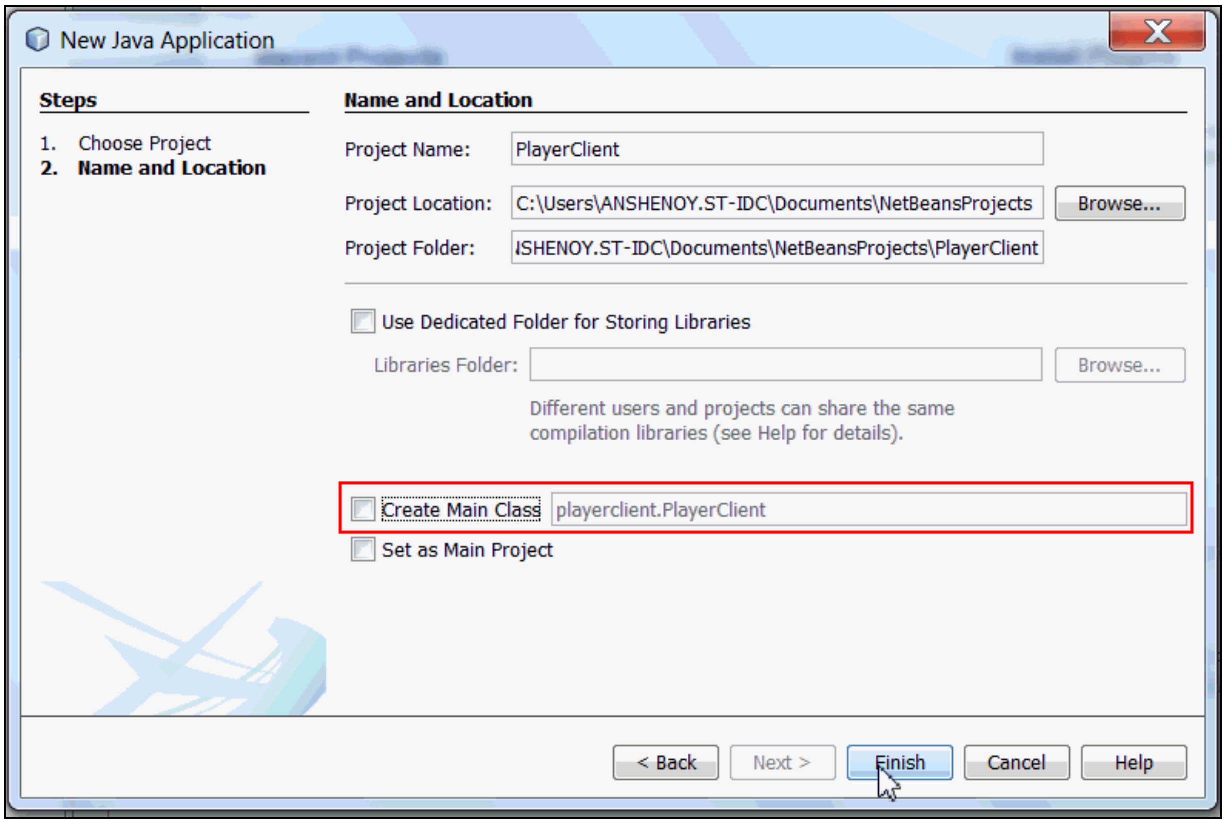
Creates a new Java SE application in a standard IDE project. You can also generate a main class in the project. Standard projects use **an IDE-generated Ant build script** to build, run, and debug your project.

4 . Perform the below steps:

a. Name the project **PlayerClient**.

b. Uncheck Create Main Class.

c. Click Finish.

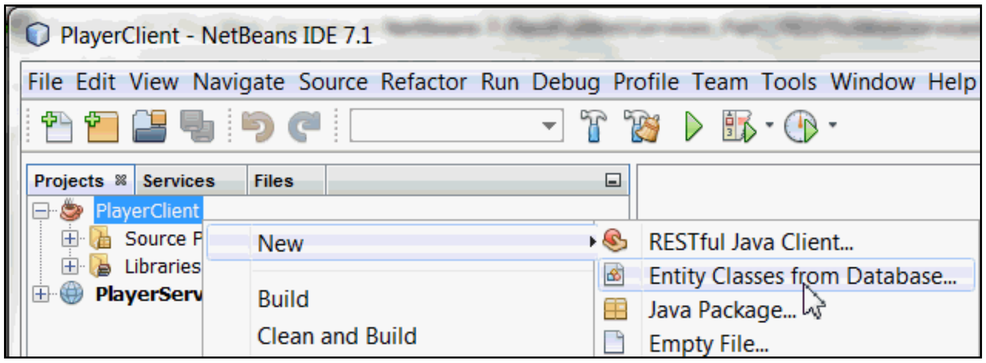


Project Name: PlayerClient
Project Location: C:\Users\ANSHENOY.ST-IDC\Documents\NetBeansProjects
Project Folder: ISHENOY.ST-IDC\Documents\NetBeansProjects\PlayerClient
Use Dedicated Folder for Storing Libraries
Libraries Folder:
Different users and projects can share the same compilation libraries (see Help for details).
Create Main Class: playerclient.PlayerClient
Set as Main Project

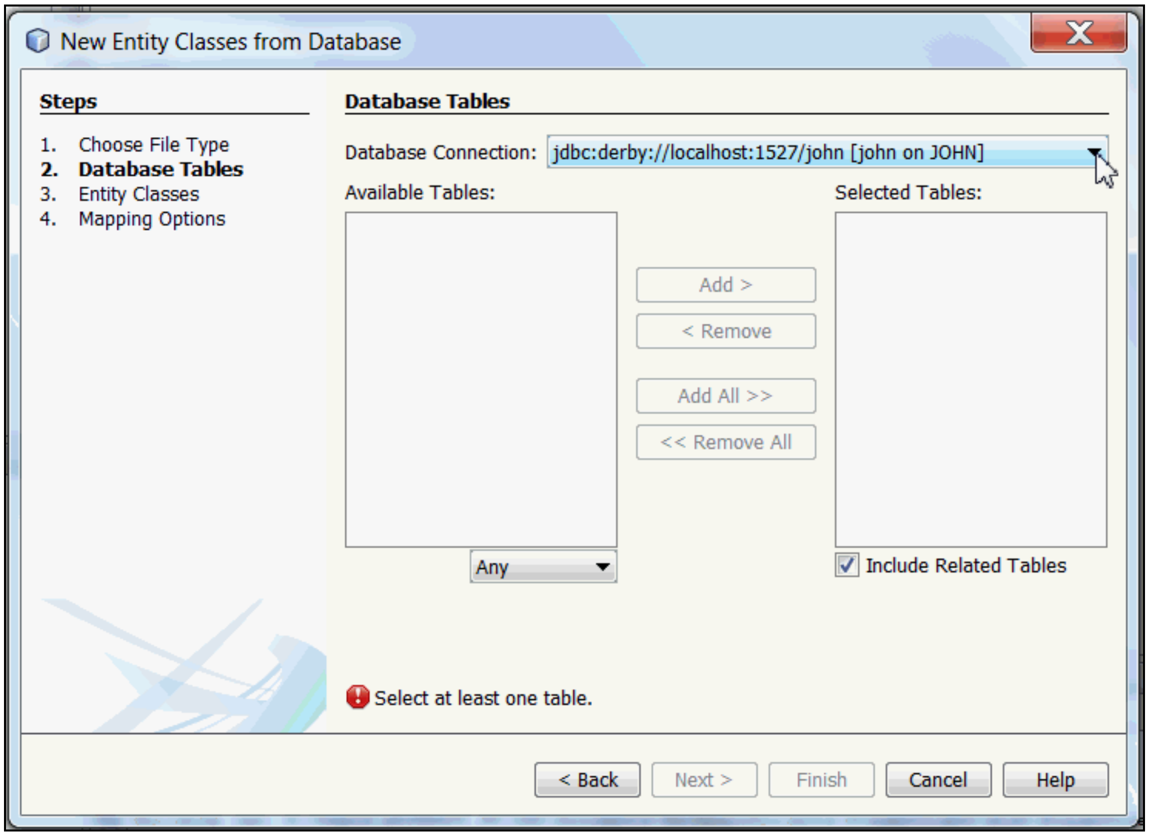
Generating Entity Classes from Database

Implementing CRUD operations on the database, requires creation of Entity Classes to communicate with the database. The following section demonstrates how to create Entity Classes from database.

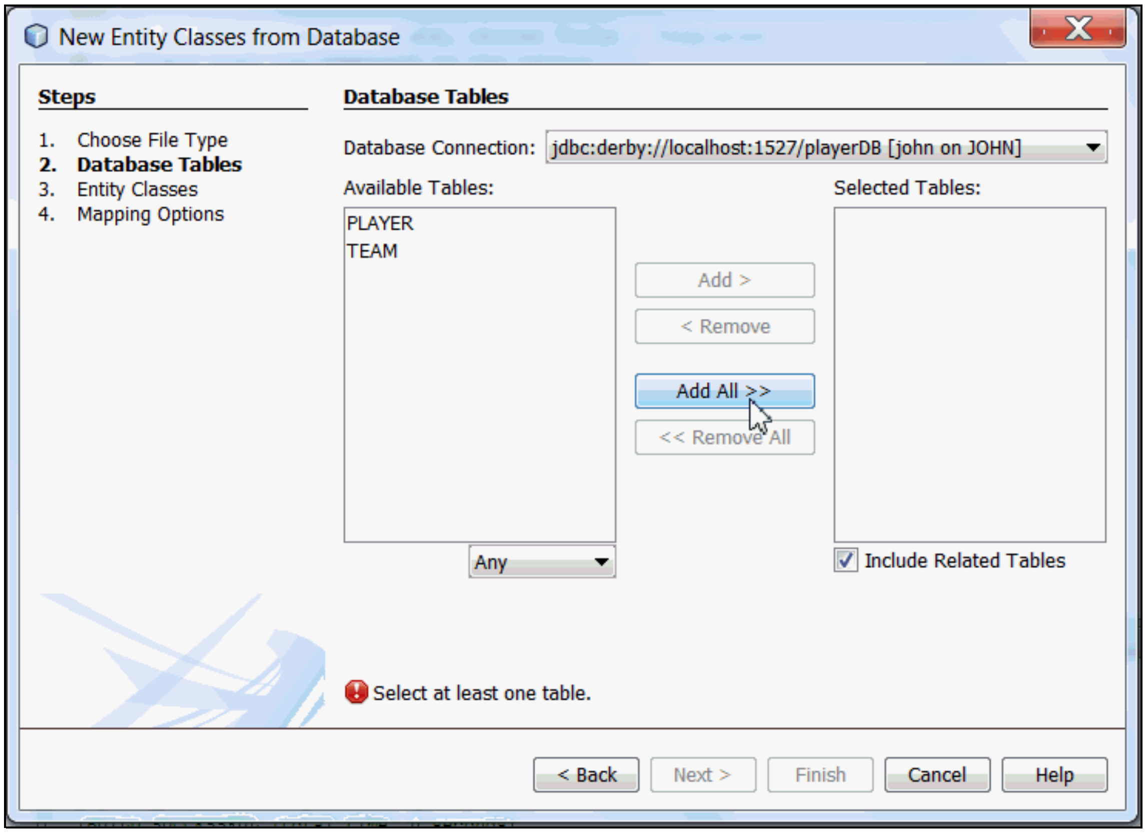
1 . Right-click **PlayerClient** Project and select ***New > Entity Classes From Database***.


PlayerClient - NetBeans IDE 7.1
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Projects
Services
Files
PlayerClient
Source P
Libraries
PlayerServ
New
Build
Clean and Build
RESTful Java Client...
Entity Classes from Database...
Java Package...
Empty File...

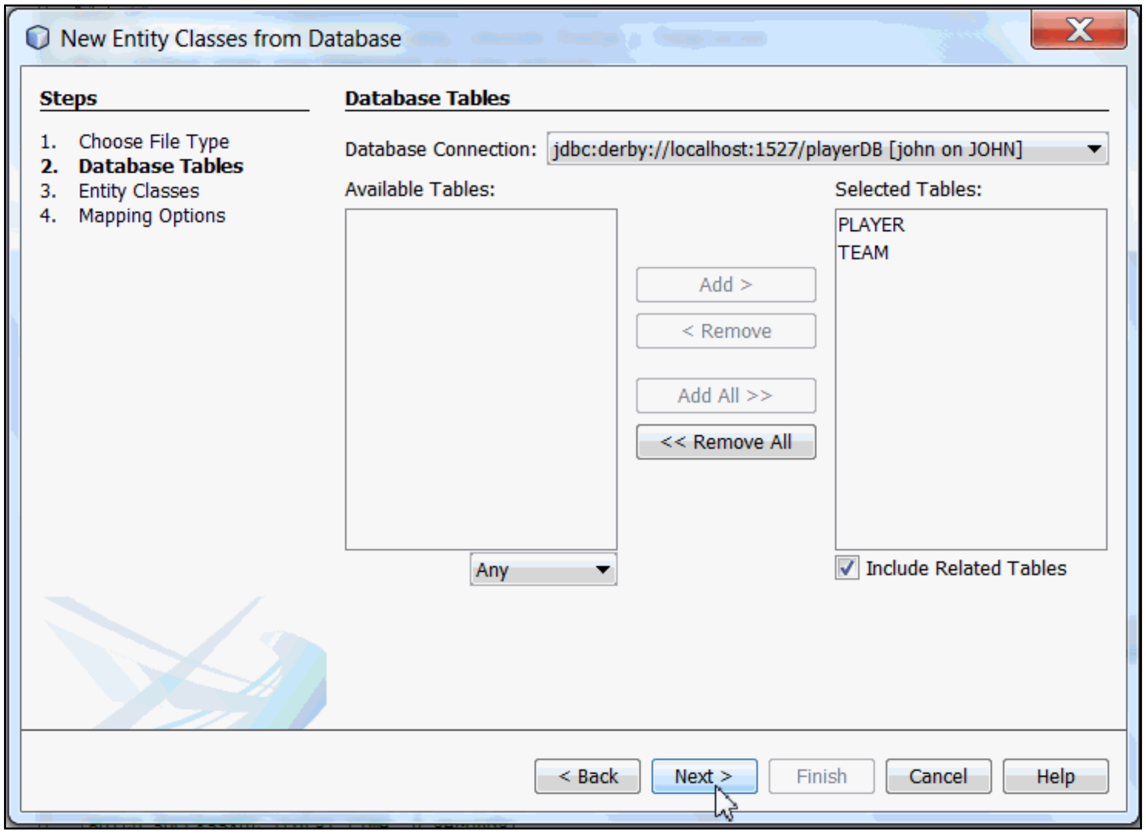
2 . a. In the Database Tables window, **Database Connection** field select `jdbc:derby://localhost:1527/playerDB[john on JOHN]` from the drop-down list.



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



You see both the tables PLAYER and TEAM in Selected Tables Category.
c. Click Next



New Entity Classes from Database

Steps

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

Database Tables

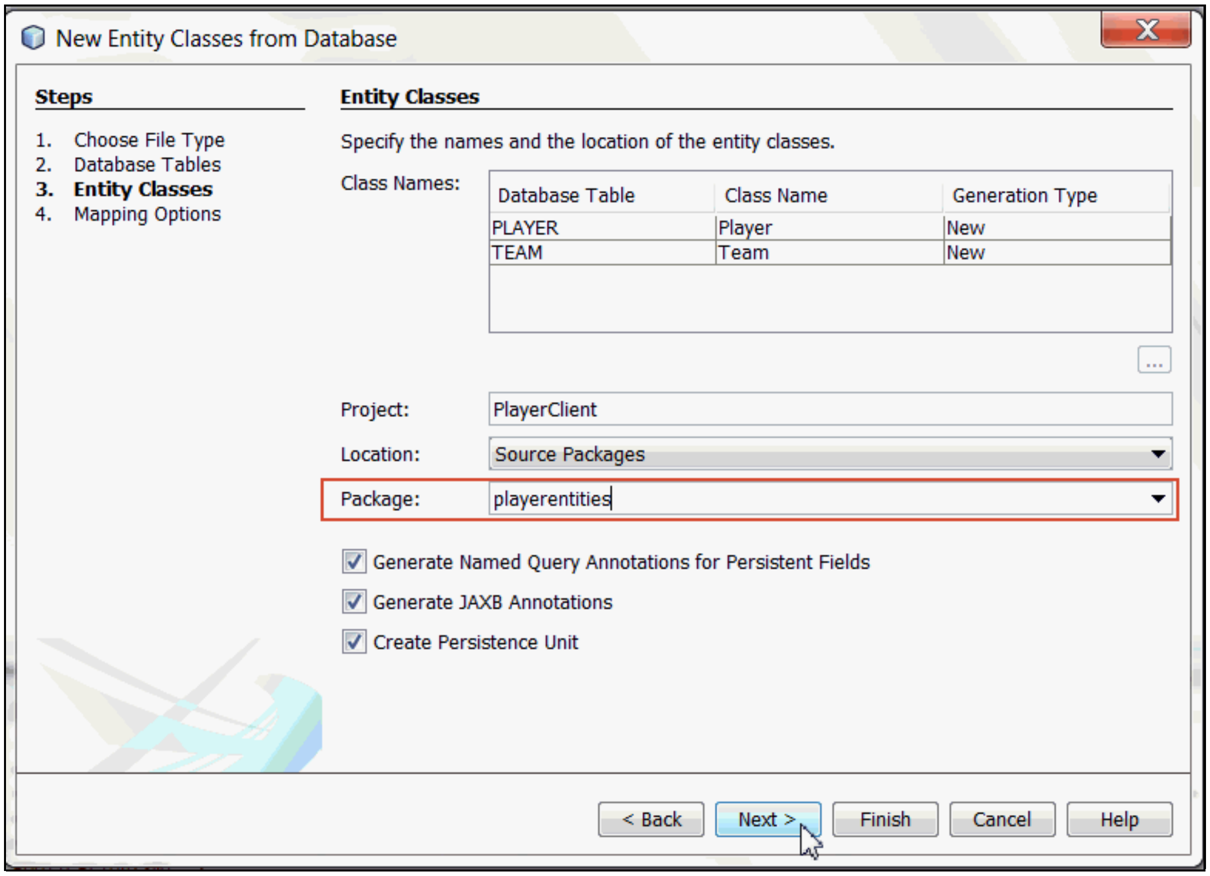
Database Connection: jdbc:derby://localhost:1527/playerDB [john on JOHN]

Available Tables:

Selected Tables:
PLAYER
TEAM

✓ Include Related Tables

3 . In the Entity classes Window, enter the Package Name as **playerentities** 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

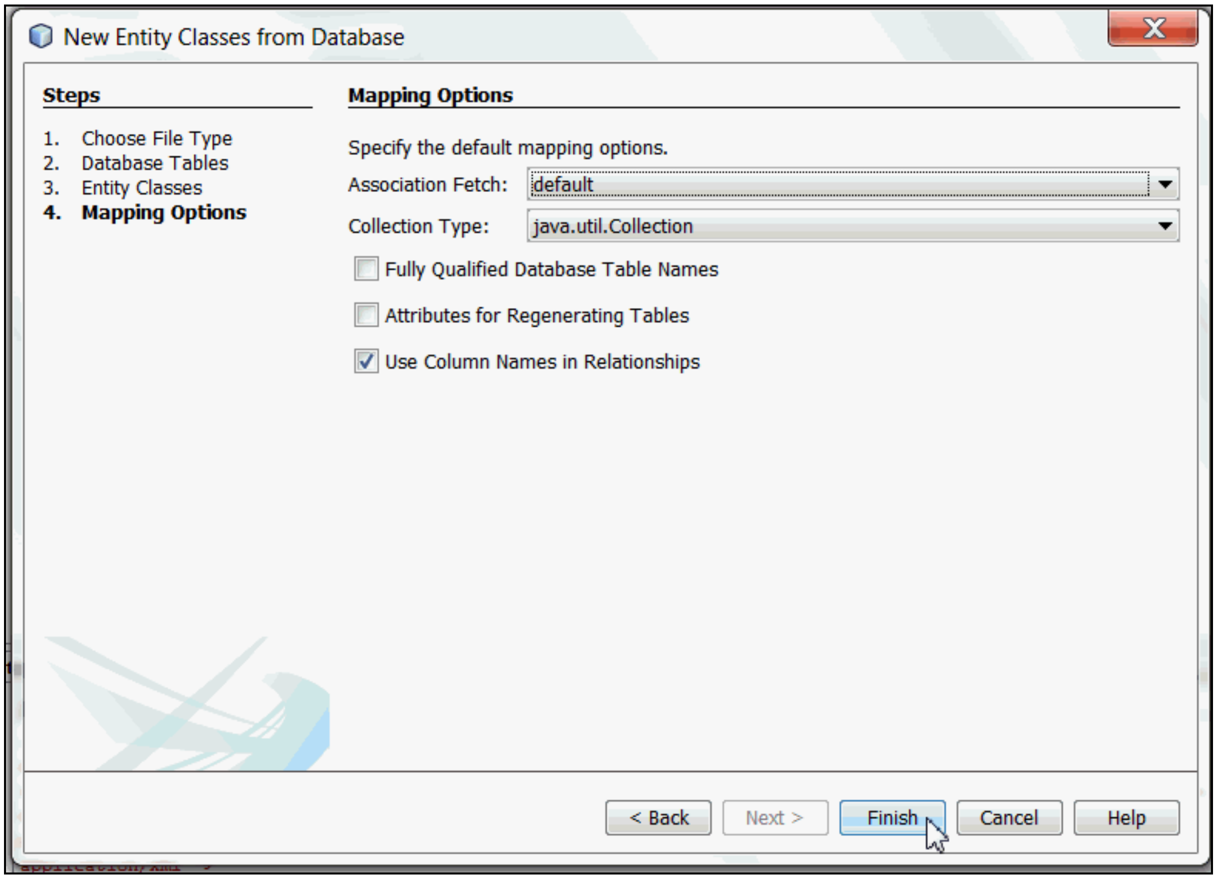
Project: PlayerClient

Location: Source Packages

Package: playerentities

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

4 . In the Mappings Options 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

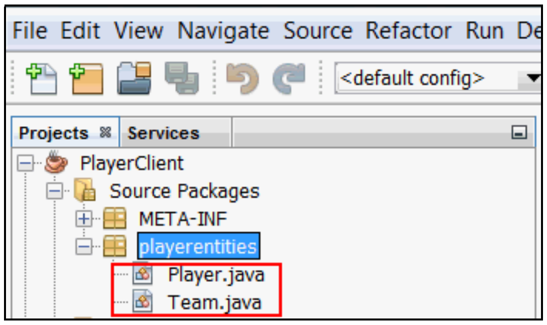
☐ Fully Qualified Database Table Names

☐ Attributes for Regenerating Tables

☑ Use Column Names in Relationships

5 . Verify the creation of Entity Classes.

a. Select the **PlayerClient** Project.
b. Expand the Source package > playerentities, you see `Team.java` and `Player.java` created.



```
PlayerClient
  Source Packages
    META-INF
    playerentities
      Player.java
      Team.java
```

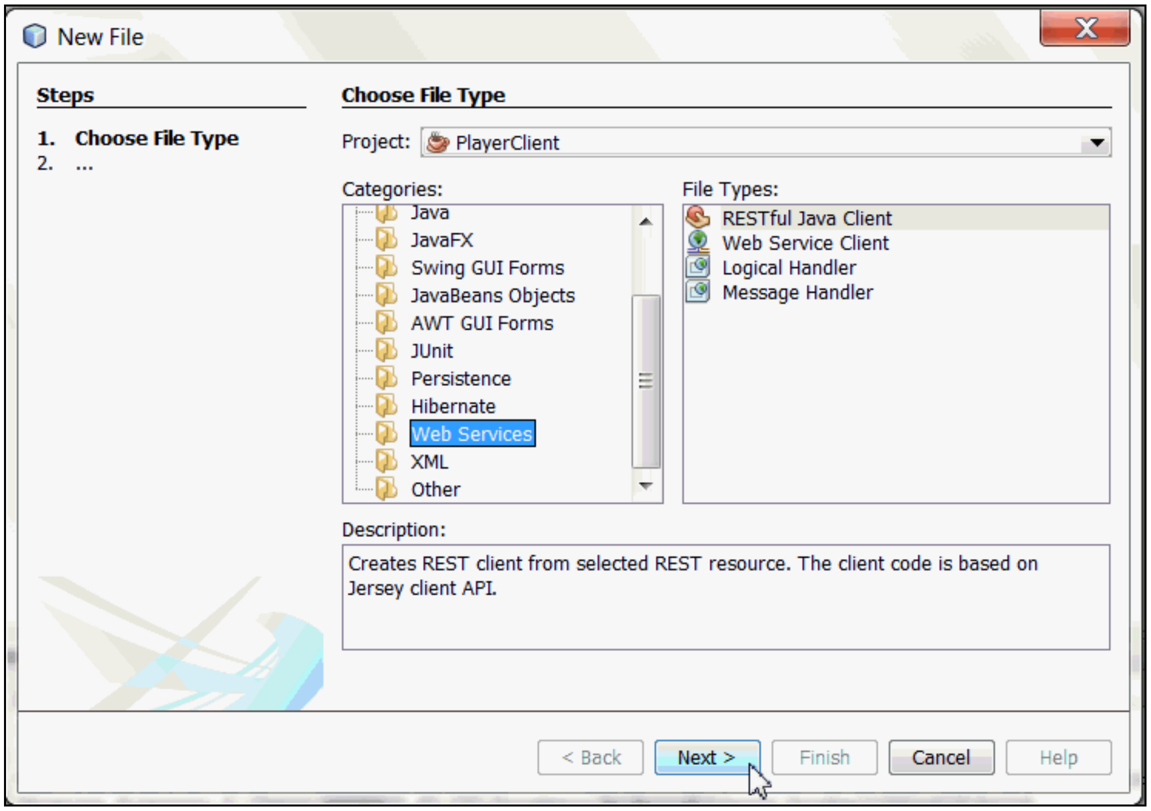
Create Operation Using RESTful Client

The following section demonstrates implemening Create operation using a RESTful Web Service. You will use RESTful Web Service to create a row in PLAYER table.

1 . Complete the following steps to create RESTful client in PlayerClient project.

a. Right-click the PlayerClient and choose **`New > Other > Web Services > RESTful Java Client`**.

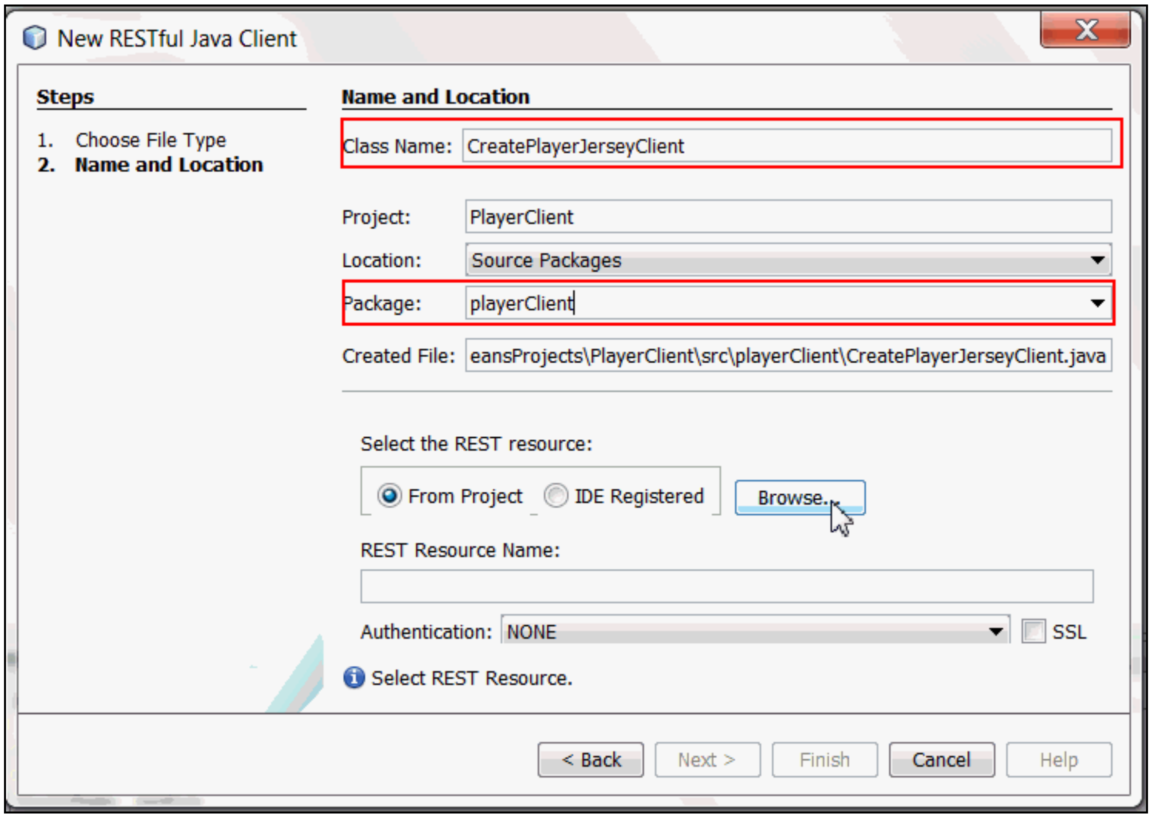
b. Click Next.



Creates REST client from selected REST resource. The client code is based on Jersey client API.

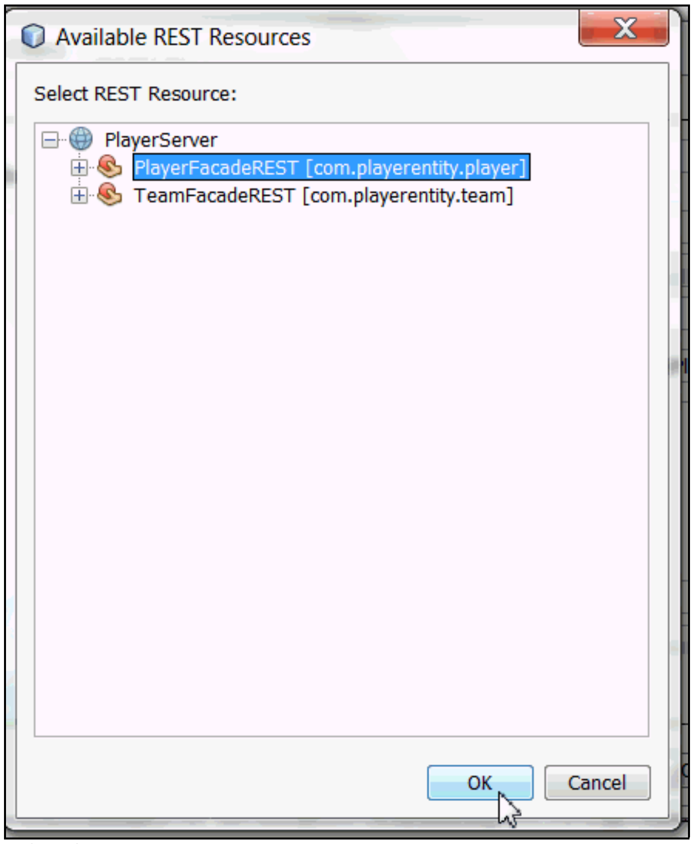
2 a. Enter the following details for the fields in the New Restful Java Client window :

Class Name: CreatePlayerJerseyClient
Package: playerClient
Select the REST resource: From Project and Click Browse.



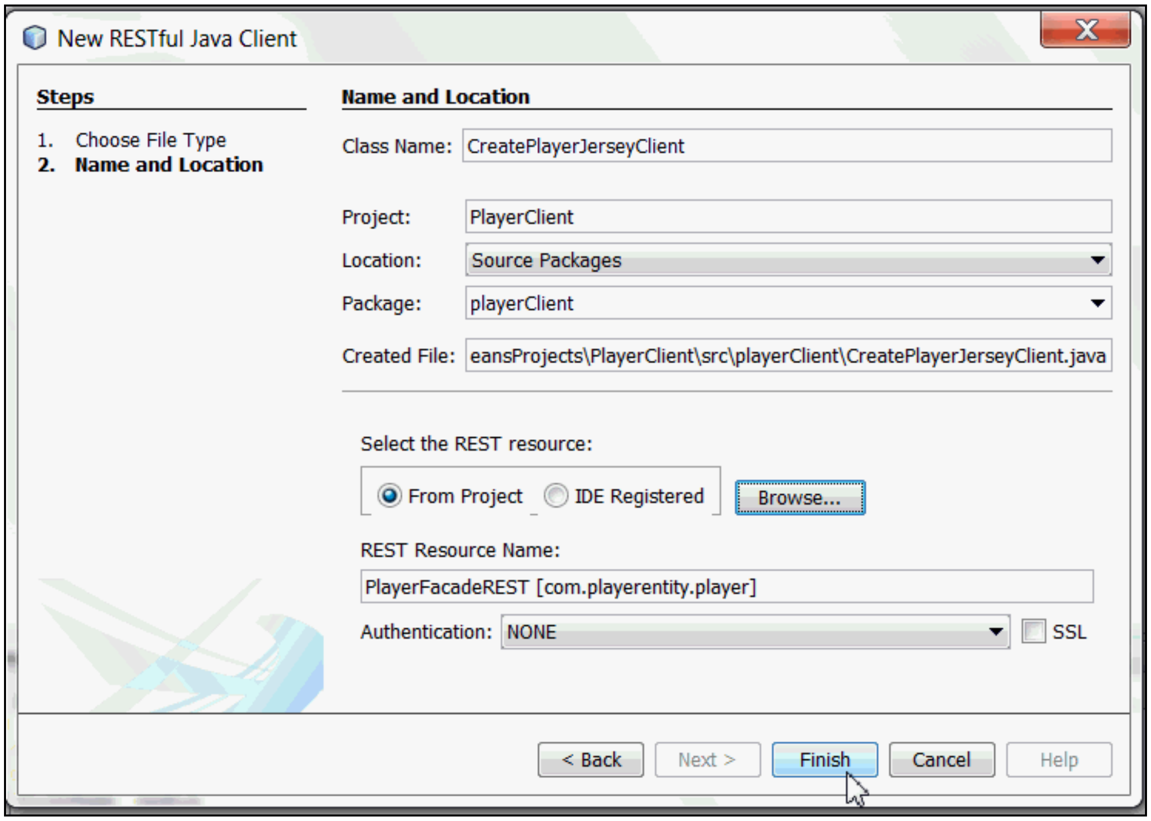
Class Name: CreatePlayerJerseyClient
Project: PlayerClient
Location: Source Packages
Package: playerClient
Created File: eansProjects\PlayerClient\src\playerClient\CreatePlayerJerseyClient.java
Authentication: NONE
Select REST Resource.

b. Select **PlayerServer**.
c. Expand the PlayerServer application.
d. Select `PlayerFacadeREST[com.playerentity.player]`.


Available REST Resources
Select REST Resource:
PlayerServer
PlayerFacadeREST [com.playerentity.player]
TeamFacadeREST [com.playerentity.team]
OK
Cancel

g. Click OK.

h. Click Finish.



Class Name: CreatePlayerJerseyClient

Project: PlayerClient

Location: Source Packages

Package: playerClient

Created File: eansProjects\PlayerClient\src\playerClient\CreatePlayerJerseyClient.java

Select the REST resource: [✓ From Project] IDE Registered Browse...

REST Resource Name: PlayerFacadeREST [com.playerentity.player]

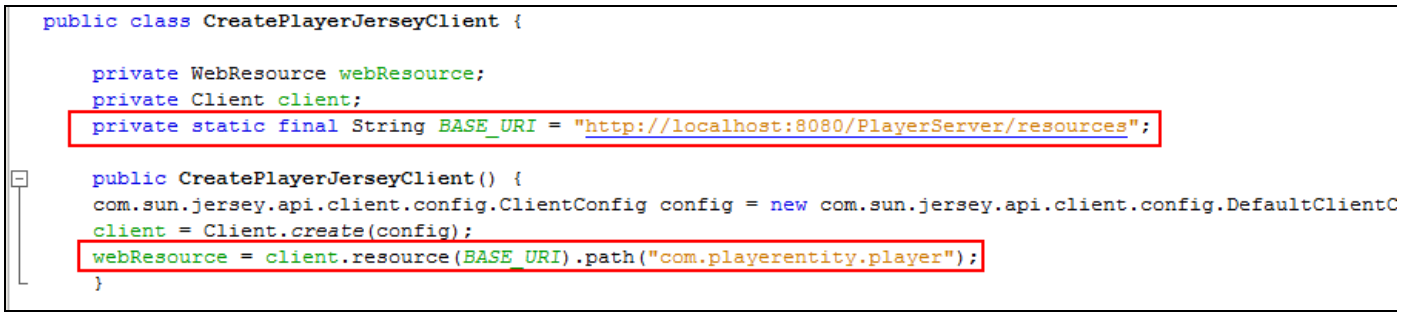
Authentication: NONE SSL

`CreatePlayerJerseyClient.java` opens in the code editor.

3 Examine `CreatePlayerJerseyClient.java`.

.

a. Identify the code snippet where the WebResource is initialized with the URI and path of the RESTful service.



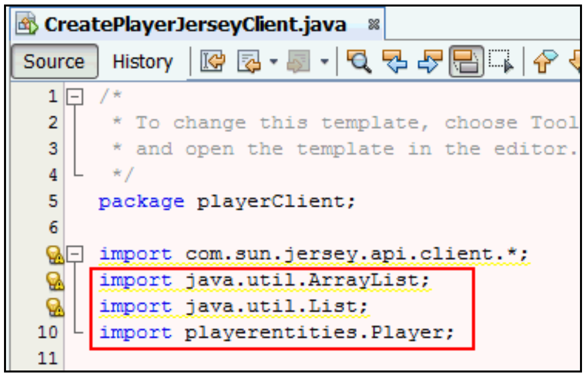
```
public class CreatePlayerJerseyClient {

    private WebResource webResource;
    private Client client;
    private static final String BASE_URI = "http://localhost:8080/PlayerServer/resources";

    public CreatePlayerJerseyClient() {
    com.sun.jersey.api.client.config.ClientConfig config = new com.sun.jersey.api.client.config.DefaultClientC
    client = Client.create(config);
    webResource = client.resource(BASE_URI).path("com.playerentity.player");
    }
```

b. Observe all the methods of the web service are present which gets, puts, deletes or posts data.
Each of the web methods has an `application/XML` and `application/JSON` format. `create_XML (Object requestEntity)` or `create_JSON(Object requestEntity)` for create operation are two examples.

4. To create a row in the **PLAYER** table using the RESTful Web Service, make following changes to `CreatePlayerJerseyClient.java`

a. Import the following packages.

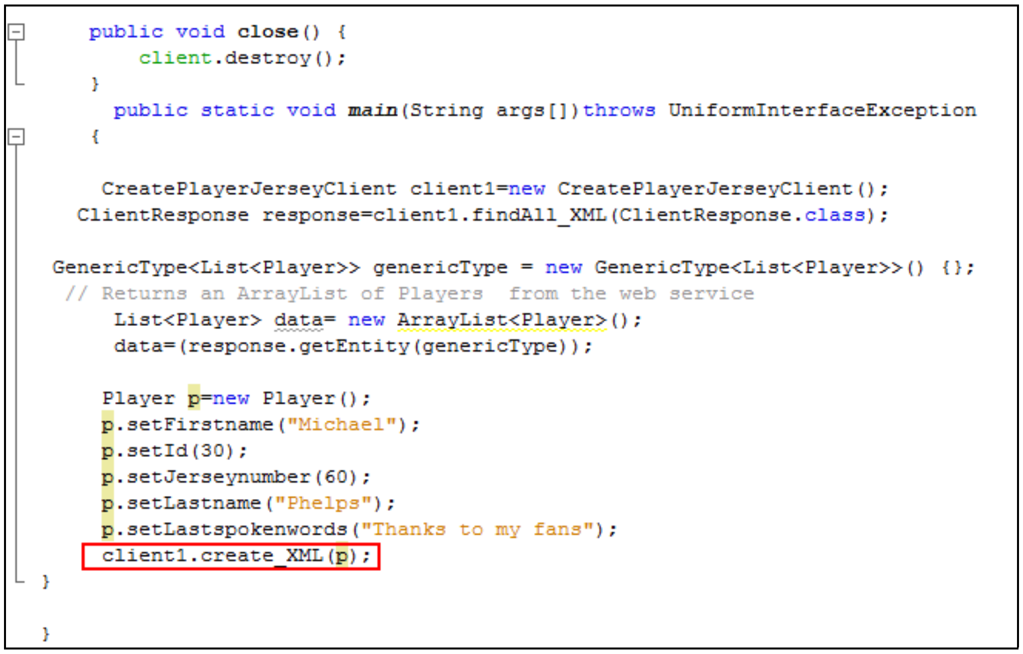
```
import java.util.ArrayList;
import java.util.List;
import playerentities.Player;
```



```
/*
 * To change this template, choose Tool
 * and open the template in the editor.
 */
package playerClient;

import com.sun.jersey.api.client.*;
import java.util.ArrayList;
import java.util.List;
import playerentities.Player;
```

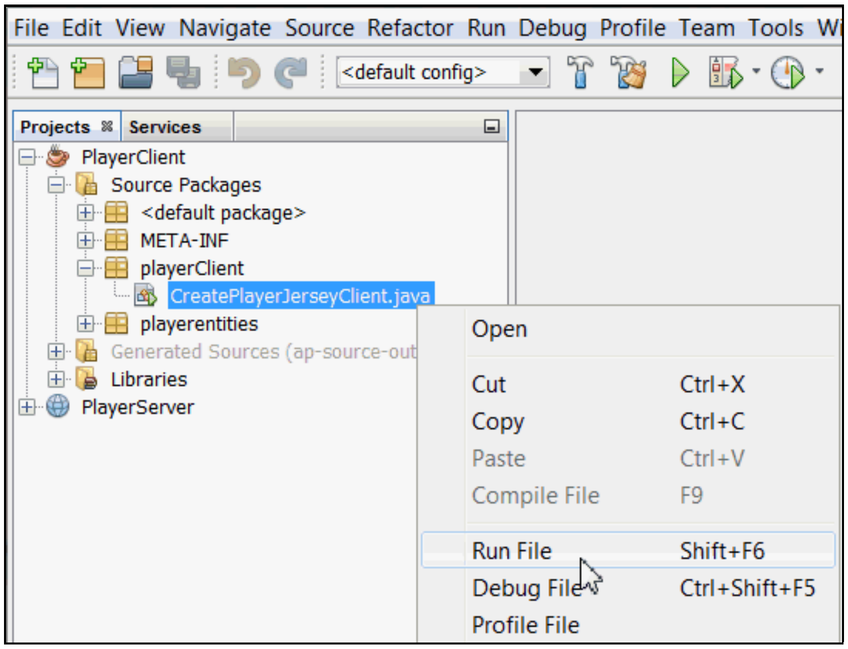
b. Add the `main` method.

```
public static void main(String args[])throws UniformInterfaceException
{

CreatePlayerJerseyClient client1=new CreatePlayerJerseyClient();
ClientResponse response=client1.findAll_XML(ClientResponse.class);

GenericType<List<Player>> genericType = new GenericType<List<Player>>() {};
// Returns an ArrayList of Players from the web service
List<Player> data= new ArrayList<Player>();
data=(response.getEntity(genericType));

Player p=new Player();
p.setFirstname("Michael");
p.setId(30);
p.setJerseynumber(60);
p.setLastname("Phelps");
p.setLastspokenwords("Thanks to my fans");
client1.create_XML(p);
}
```



```
    public void close() {
        client.destroy();
    }
     public static void main(String args[])throws UniformInterfaceException
    {

     CreatePlayerJerseyClient client1=new CreatePlayerJerseyClient();
   ClientResponse response=client1.findAll_XML(ClientResponse.class);

 GenericType<List<Player>> genericType = new GenericType<List<Player>>() {};
  // Returns an ArrayList of Players  from the web service
      List<Player> data= new ArrayList<Player>();
      data=(response.getEntity(genericType));

     Player p=new Player();
     p.setFirstname("Michael");
     p.setId(30);
     p.setJerseynumber(60);
     p.setLastname("Phelps");
     p.setLastspokenwords("Thanks to my fans");
     client1.create_XML(p);
 }

 }
```

5 In the Projects window, right-click `CreatePlayerJerseyClient.java` and select **Run File.**

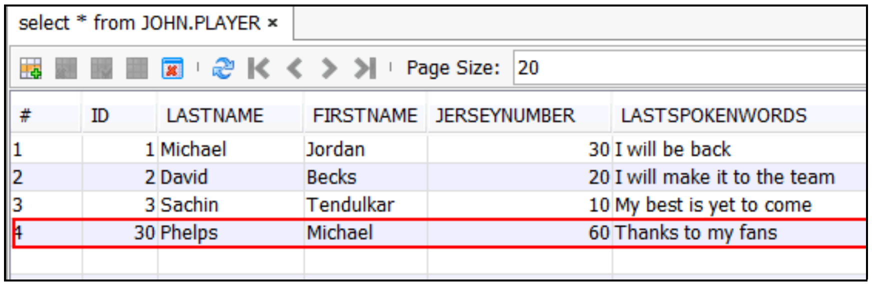


6 Verify the output. It can be done in two ways: examine the contents of the PLAYER table or Generate Web Services test client.

a. To examine the contents of the PLAYER table, complete the below steps:

1. In the **Services** window, expand the `jdbc:derby://localhost:1527/playerDB` connection under the Databases node.
2. Right-click the connection and select Refresh.
3. Expand the JOHN schema > Expand Tables Node >PLAYER Table.
4. Right-click PLAYER table node and select **View Data**.

5. Examine the PLAYER table data, a new row would be created with the details specified for Player object p.

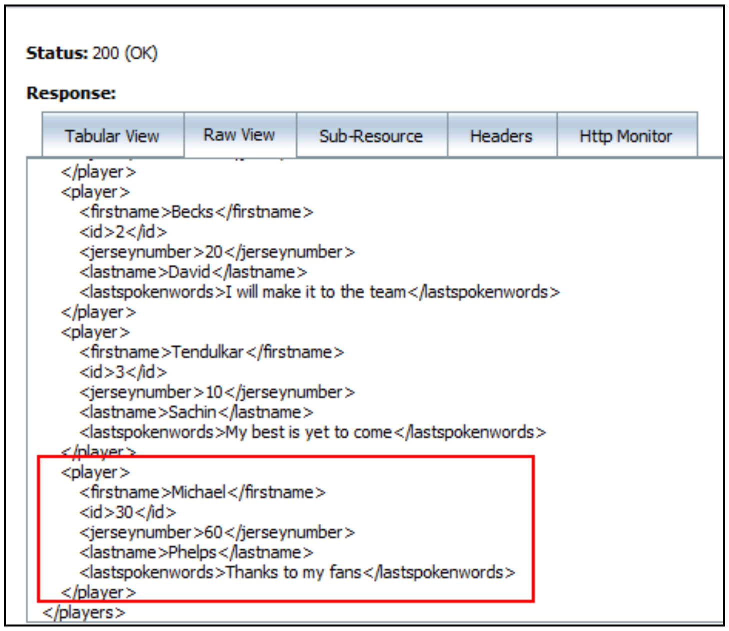

select * from JOHN.PLAYER

Page Size: 20

#	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	30	Phelps	Michael	60	Thanks to my fans

b. Generate Web Services test client as explained in **Part-1**. After the test client is deployed to test the Web Services Client, complete the following steps.

1. Select one resource node, `com.playerentity.player`.
2. In the "Choose method to test" field, select GET (application/xml).
3. Click Test.
4. Examine the output, response to an `application/xml` request.You will see the newly created Player object in the XML file.


Status: 200 (OK)

Response:

```
    </player>
    <player>
        <firstname>Becks</firstname>
        <id>2</id>
        <jerseynumber>20</jerseynumber>
        <lastname>David</lastname>
        <lastspokenwords>I will make it to the team</lastspokenwords>
    </player>
    <player>
        <firstname>Tendulkar</firstname>
        <id>3</id>
        <jerseynumber>10</jerseynumber>
        <lastname>Sachin</lastname>
        <lastspokenwords>My best is yet to come</lastspokenwords>
    </player>
    <player>
        <firstname>Michael</firstname>
        <id>30</id>
        <jerseynumber>60</jerseynumber>
        <lastname>Phelps</lastname>
        <lastspokenwords>Thanks to my fans</lastspokenwords>
    </player>
</players>
```

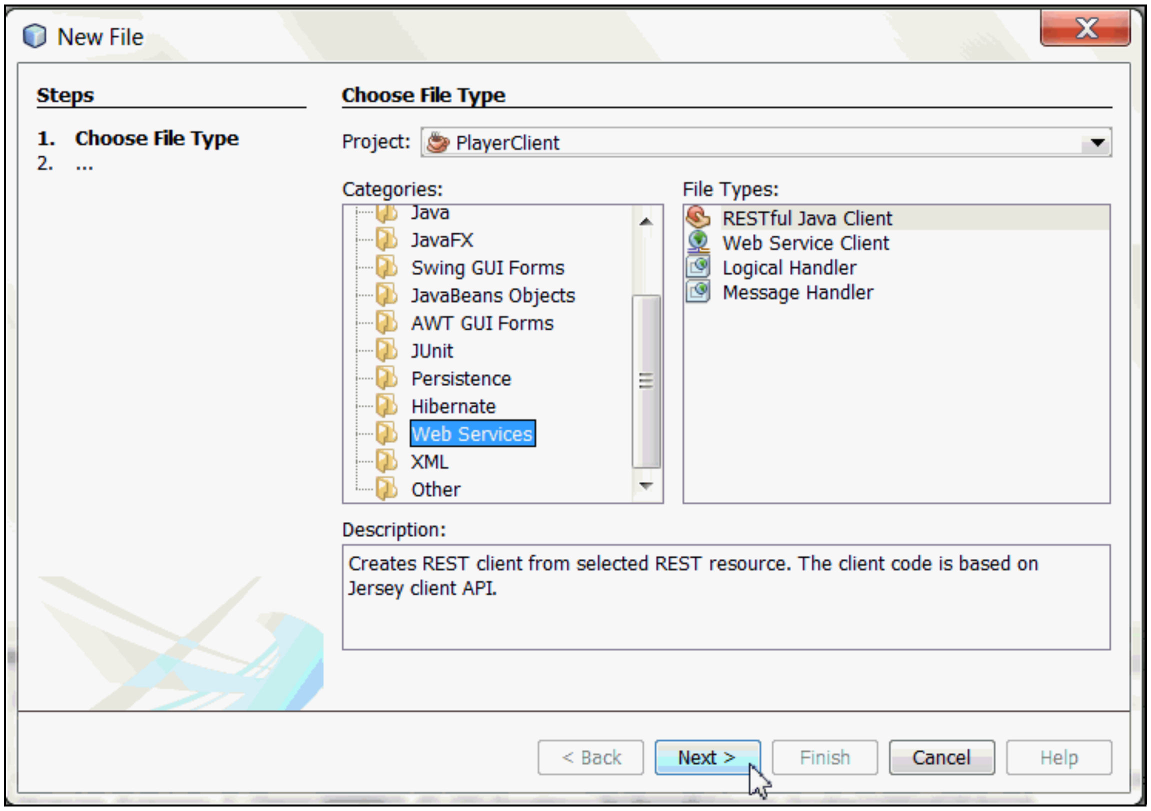
Retrieve Operation Using RESTful Client

The following section demonstrates how to implement a Retrieve operation on the database using RESTful Web Service. You will use RESTful Web Service to retrieve all the rows in the PLAYER table and display in the console.

1. Complete the following steps to create RESTful client in PlayerClient project.

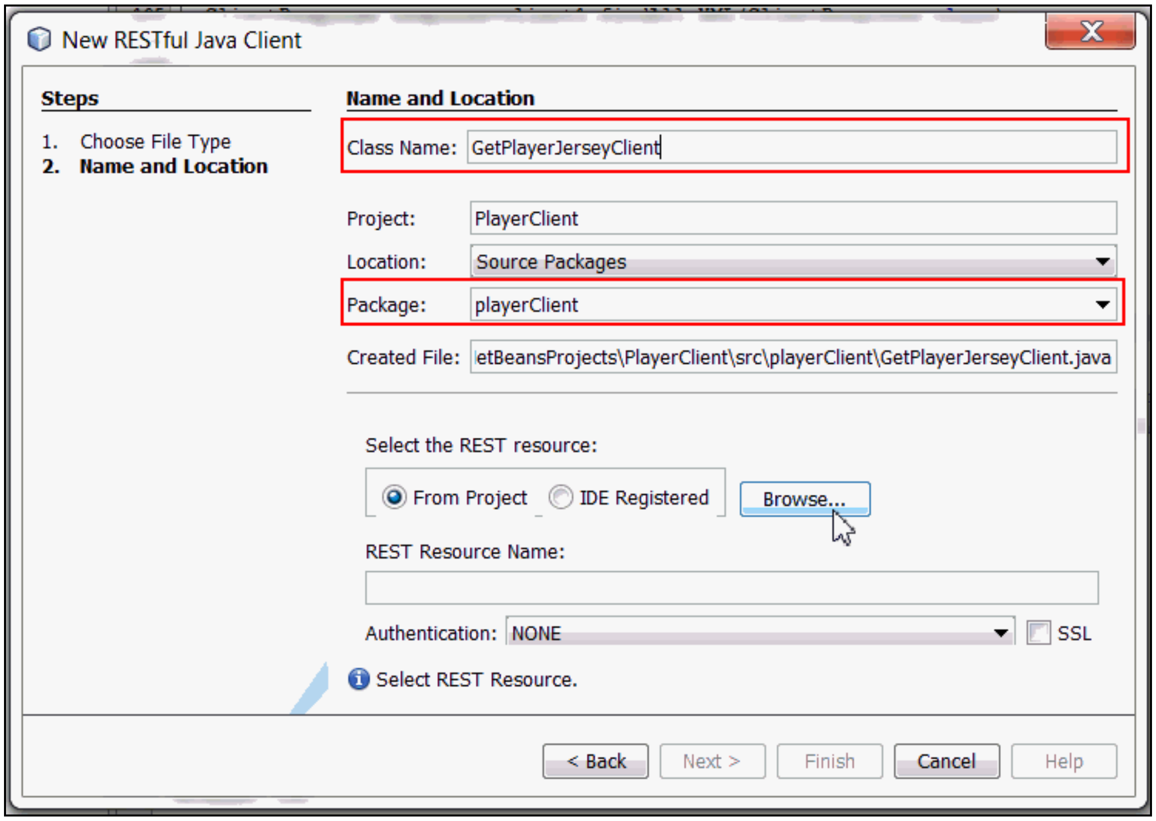
a. Right-click the PlayerClient and choose `New > Other > Web Services > RESTful Java Client`.

b. Click Next.



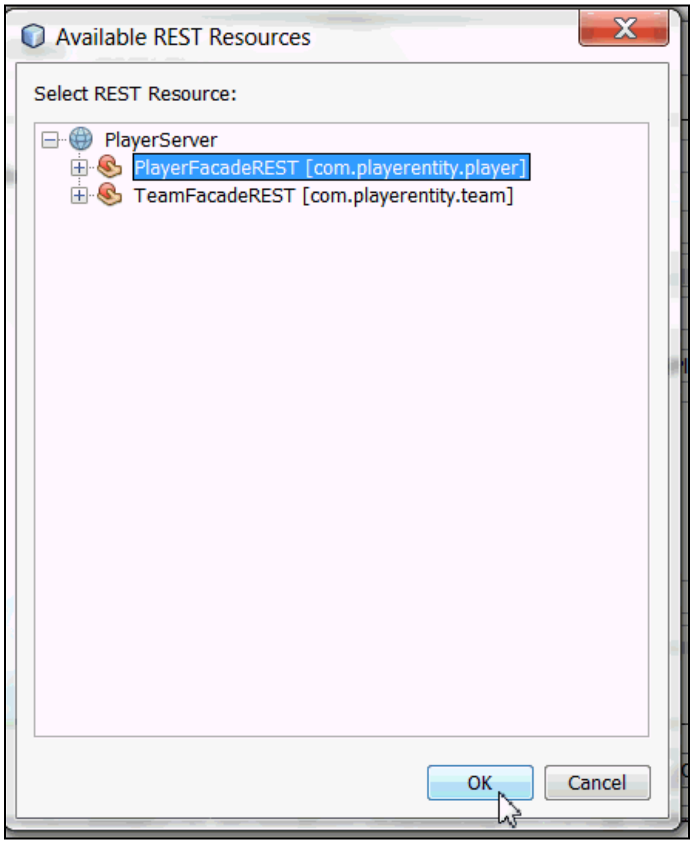
Creates REST client from selected REST resource. The client code is based on Jersey client API.

2. Enter the following details for the fields in the New Restful Java Client window :
   a. **Class Name:** GetPlayerJerseyClient
   b. **Package:** playerClient
   c. **Select the REST resource:** From Project and Click Browse.

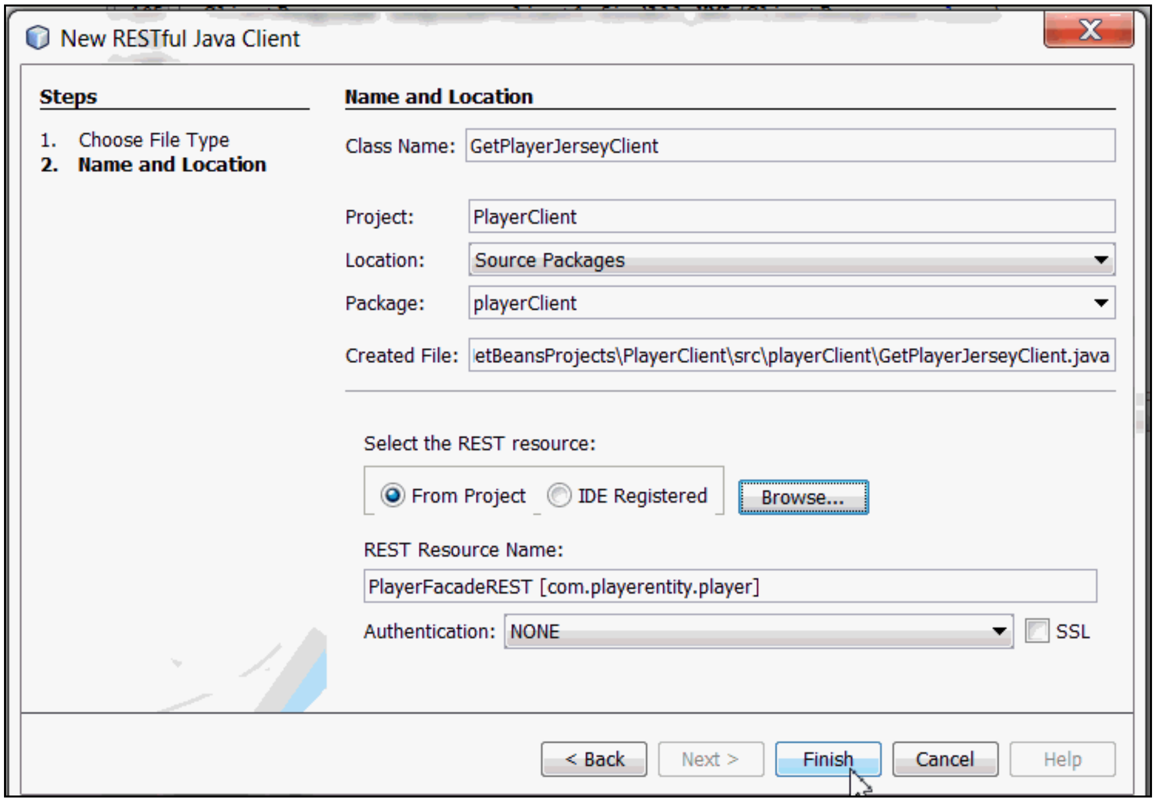


Class Name: GetPlayerJerseyClient
Project: PlayerClient
Location: Source Packages
Package: playerClient
Created File: etBeansProjects\PlayerClient\src\playerClient\GetPlayerJerseyClient.java
Authentication: NONE
Select REST Resource.

d. Select PlayerServer
e. Expand the **PlayerServer** application.
f. Select `PlayerFacadeREST[com.playerentity.player]`
g. Click OK.


Available REST Resources
Select REST Resource:
PlayerServer
PlayerFacadeREST [com.playerentity.player]
TeamFacadeREST [com.playerentity.team]
OK
Cancel

h. Click Finish.



Class Name: GetPlayerJerseyClient

Project: PlayerClient

Location: Source Packages

Package: playerClient

Created File: etBeansProjects\PlayerClient\src\playerClient\GetPlayerJerseyClient.java

Select the REST resource: [✓ From Project] IDE Registered

REST Resource Name: PlayerFacadeREST [com.playerentity.player]

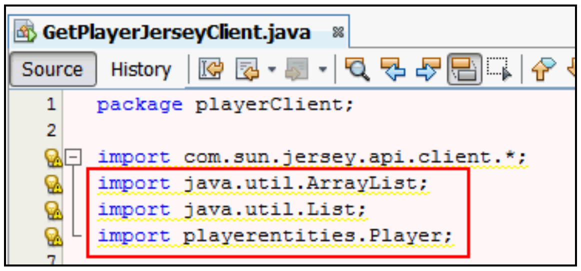
Authentication: NONE SSL

`GetPlayerJerseyClient.java` opens in the code editor.

3 . To retrieve the contents of the Player table using the RESTful Web Service, make following changes to `GetPlayerJerseyClient.java`

a. Import the following packages.

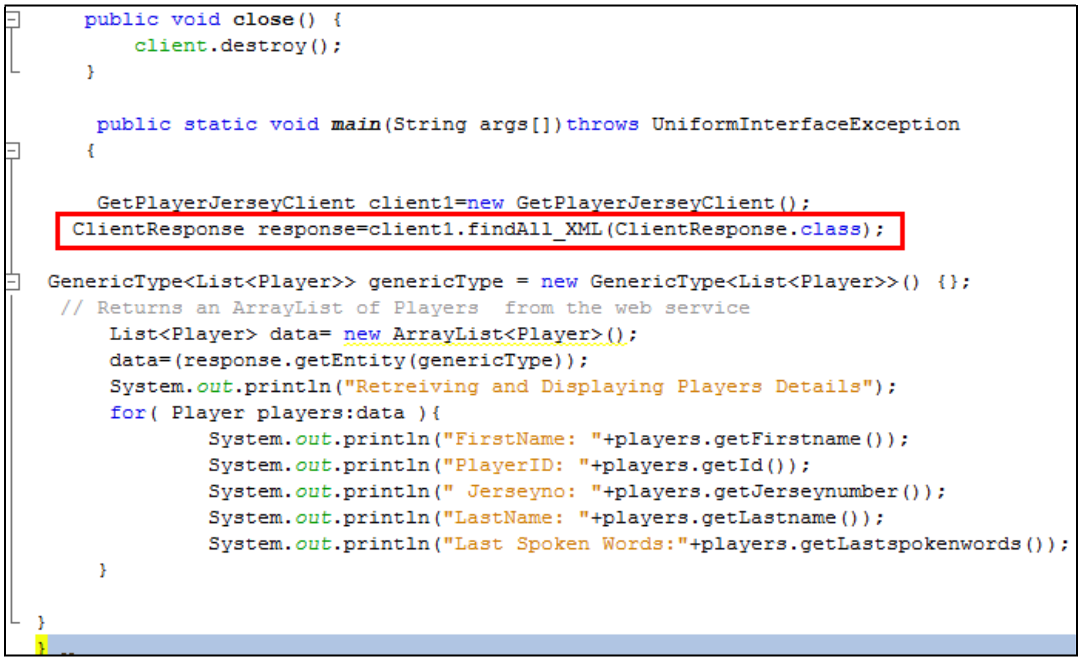
```
import java.util.ArrayList;
import java.util.List;
import playerentities.Player;
```



```
package playerClient;

import com.sun.jersey.api.client.*;
import java.util.ArrayList;
import java.util.List;
import playerentities.Player;
```

b. Add the `main` method.

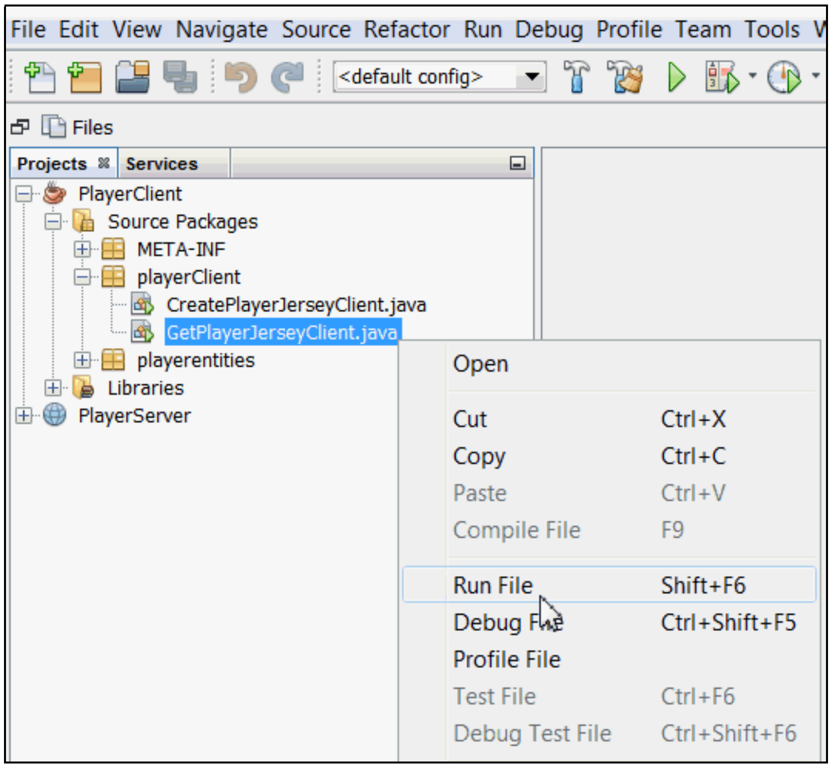
```
public static void main(String args[])throws UniformInterfaceException
{

GetPlayerJerseyClient client1=new GetPlayerJerseyClient();
ClientResponse response=client1.findAll_XML(ClientResponse.class);


GenericType<List<Player>> genericType = new GenericType<List<Player>>() {};
// Returns an ArrayList of Players from the web service
List<Player> data= new ArrayList<Player>();
data=(response.getEntity(genericType));
System.out.println("Retreiving and Displaying Players Details");
for( Player players:data ){
System.out.println("FirstName: "+players.getFirstname());
System.out.println("PlayerID: "+players.getId());
System.out.println(" Jerseyno: "+players.getJerseynumber());
System.out.println("LastName: "+players.getLastname());
System.out.println("Last Spoken Words:"+players.getLastspokenwords());
}

}
```



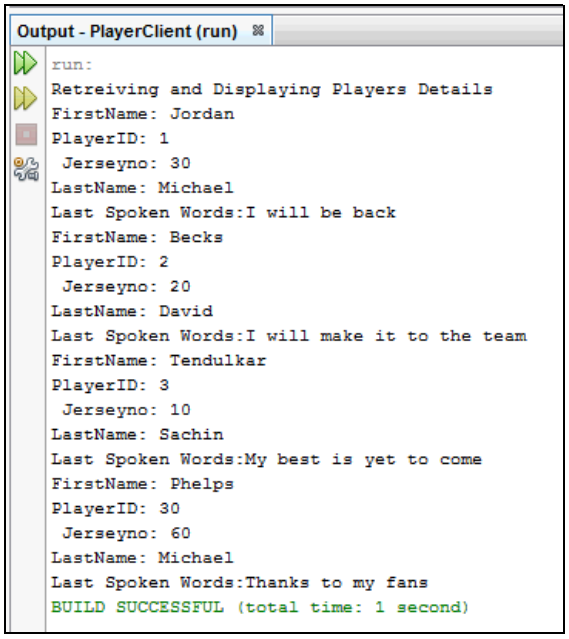
```
    public void close() {
        client.destroy();
    }

     public static void main(String args[])throws UniformInterfaceException
    {

     GetPlayerJerseyClient client1=new GetPlayerJerseyClient();
   ClientResponse response=client1.findAll_XML(ClientResponse.class);

  GenericType<List<Player>> genericType = new GenericType<List<Player>>() {};
   // Returns an ArrayList of Players  from the web service
      List<Player> data= new ArrayList<Player>();
      data=(response.getEntity(genericType));
      System.out.println("Retreiving and Displaying Players Details");
      for( Player players:data ){
              System.out.println("FirstName:  "+players.getFirstname());
              System.out.println("PlayerID:  "+players.getId());
              System.out.println(" Jerseyno:  "+players.getJerseynumber());
              System.out.println("LastName:  "+players.getLastname());
              System.out.println("Last Spoken Words:"+players.getLastspokenwords());
      }

  }
}
```

Examine `findAll_XML()` method that is used to retrieve the records from the PLAYER table.

4 . In the Projects window, right-click `GetPlayerJerseyClient.java` and select **Run File.**



5 . Examine the output, PLAYER table content is displayed in the console.



```
run:
Retreiving and Displaying Players Details
FirstName: Jordan
PlayerID: 1
 Jerseyno: 30
LastName: Michael
Last Spoken Words:I will be back
FirstName: Becks
PlayerID: 2
 Jerseyno: 20
LastName: David
Last Spoken Words:I will make it to the team
FirstName: Tendulkar
PlayerID: 3
 Jerseyno: 10
LastName: Sachin
Last Spoken Words:My best is yet to come
FirstName: Phelps
PlayerID: 30
 Jerseyno: 60
LastName: Michael
Last Spoken Words:Thanks to my fans
BUILD SUCCESSFUL (total time: 1 second)
```

Update Operation Using RESTful Client

The following section demonstrates how to implement Update operation on the database using RESTful Web Service. You will use RESTful Web Service to update a specified row in the PLAYER table.

1 . To update a particular row in the PLAYER table using the RESTful Web Service, complete the following step:

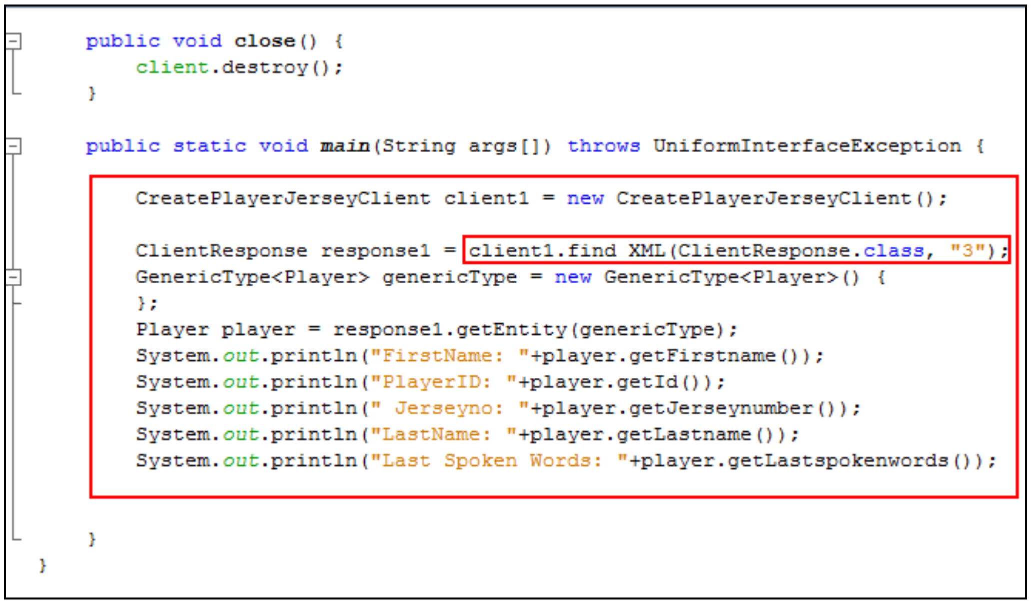
Make the following changes to main method in `CreatePlayerJerseyClient.java.`

a. Delete the contents of the `main` method.

b. Add the below lines of code to the `main` method :

```
CreatePlayerJerseyClient client1=new CreatePlayerJerseyClient();
```

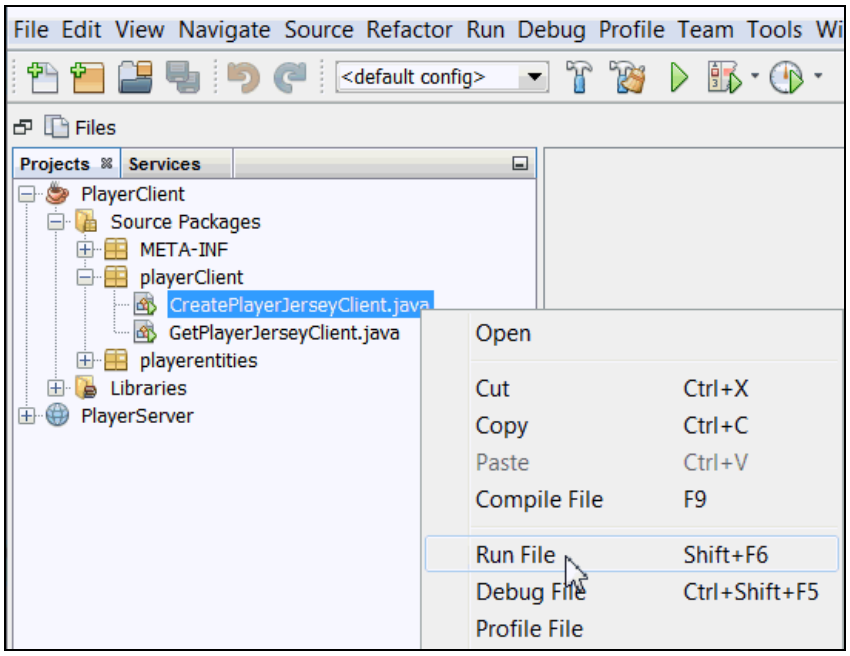
```
ClientResponse response1= client1.find_XML(ClientResponse.class,"3");
GenericType<Player> genericType = new GenericType<Player>() {};
Player player=response1.getEntity(genericType);
System.out.println("FirstName: "+player.getFirstname());
System.out.println("PlayerID: "+player.getId());
System.out.println(" Jerseyno: "+player.getJerseynumber());
System.out.println("LastName: "+player.getLastname());
System.out.println("Last Spoken Words: "+player.getLastspokenwords());
```



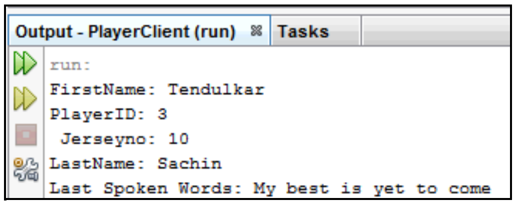
```
    public void close() {
        client.destroy();
    }

    public static void main(String args[]) throws UniformInterfaceException {
        CreatePlayerJerseyClient client1 = new CreatePlayerJerseyClient();

        ClientResponse response1 = client1.find_XML(ClientResponse.class, "3");
        GenericType<Player> genericType = new GenericType<Player>() {
        };
        Player player = response1.getEntity(genericType);
        System.out.println("FirstName: "+player.getFirstname());
        System.out.println("PlayerID: "+player.getId());
        System.out.println(" Jerseyno: "+player.getJerseynumber());
        System.out.println("LastName: "+player.getLastname());
        System.out.println("Last Spoken Words: "+player.getLastspokenwords());

    }
}
```

The above code retrieves Player with id equals 3 from the Player Table. The `find_XML()` method is used to retrieve the Player object with the specified Player id .

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



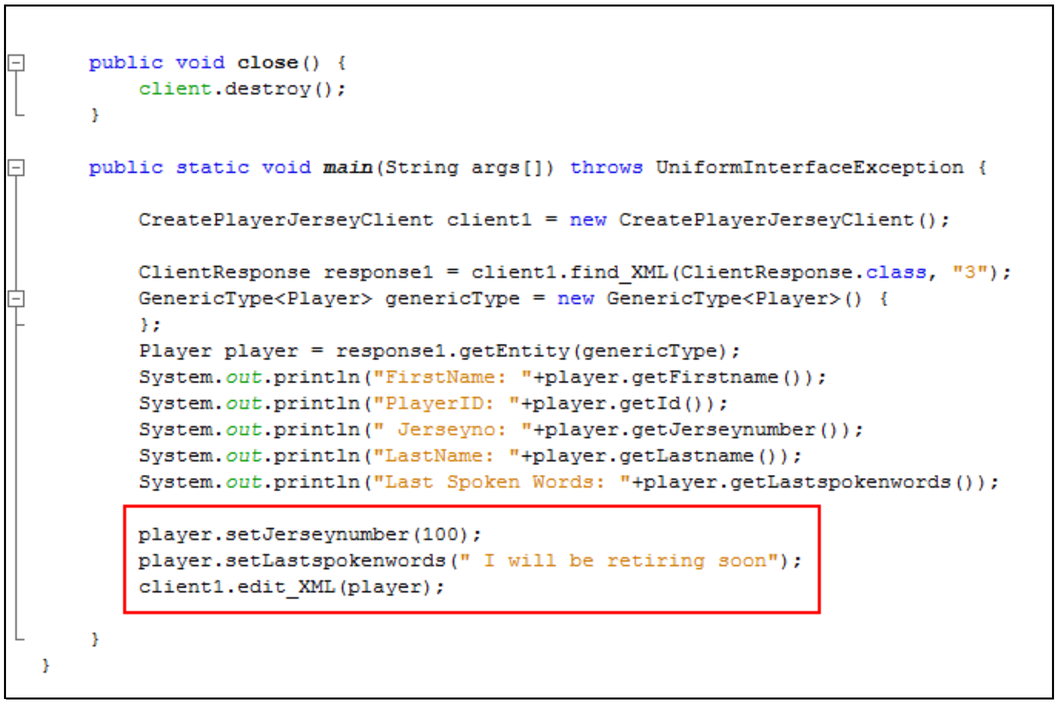
3 . Verify the output, the details Player with id 3 is displayed on the console.



```
run:
FirstName: Tendulkar
PlayerID: 3
 Jerseyno: 10
LastName: Sachin
Last Spoken Words: My best is yet to come
```

4 . Add the below code to the `main` method to update the retrieved Player object.

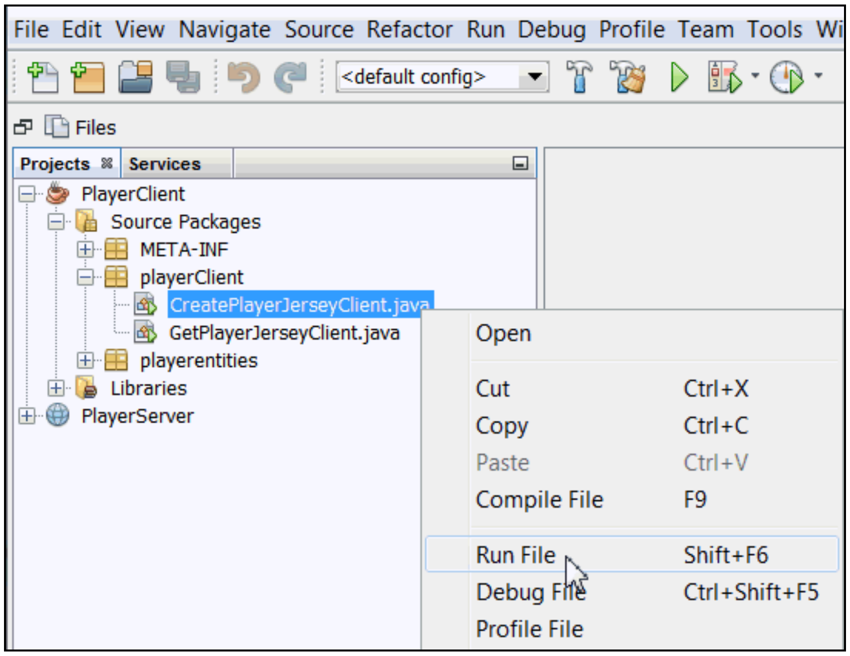
```
player.setJerseynumber(100);
player.setLastspokenwords(" I will be retiring soon");
client1.edit_XML(player);
```



```
    public void close() {
        client.destroy();
    }

    public static void main(String args[]) throws UniformInterfaceException {

        CreatePlayerJerseyClient client1 = new CreatePlayerJerseyClient();

        ClientResponse response1 = client1.find_XML(ClientResponse.class, "3");
        GenericType<Player> genericType = new GenericType<Player>() {
        };
        Player player = response1.getEntity(genericType);
        System.out.println("FirstName: "+player.getFirstname());
        System.out.println("PlayerID: "+player.getId());
        System.out.println(" Jerseyno: "+player.getJerseynumber());
        System.out.println("LastName: "+player.getLastname());
        System.out.println("Last Spoken Words: "+player.getLastspokenwords());

        player.setJerseynumber(100);
        player.setLastspokenwords(" I will be retiring soon");
        client1.edit_XML(player);

    }
}
```

The above code updates the following fields of the row with player id equals 3 in the PLAYER table : `JerseyNumber` to `100` and `LastSpokenWords` to "`I will be retiring soon`". The `edit_XML()` method is used to update the record in the PLAYER table.

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



6 . Verify the output. It can be done in two ways: examine the contents of the PLAYER table or Generate Web Services test client .

a. To examine the contents of the PLAYER table, complete the following steps:

1. In the **Services** window, expand the `jdbc:derby://localhost:1527/playerDB` connection under the Databases node.
2. Right-click the connection and select Refresh.
3. Expand the JOHN schema > Expand Tables Node >PLAYER Table.
4. Right-click PLAYER table node and select **View Data**.

5. The two fields of the Player Table - `Jerseynumber` and `LastSpokenWords` of row with `PlayerID` 3 are updated.

b.Generate Web Services test client as explained in **Part-1**. After the test client is deployed to test the Web Services Client, complete the following steps.

1. Select one resource node, `com.playerentity.player`.
2. In the "Choose method to test" field, select GET (application/xml).
3. Click Test.
4. Examine the output, response to an `application/xml` request.You will see the updated Player object with playerID 3 in the XML file.

Delete Operation Using RESTful Client

The following section demonstrates how to implement a Delete operation on the database using RESTful Web Service. You will use RESTful Web Service to delete a specified row in the PLAYER table.

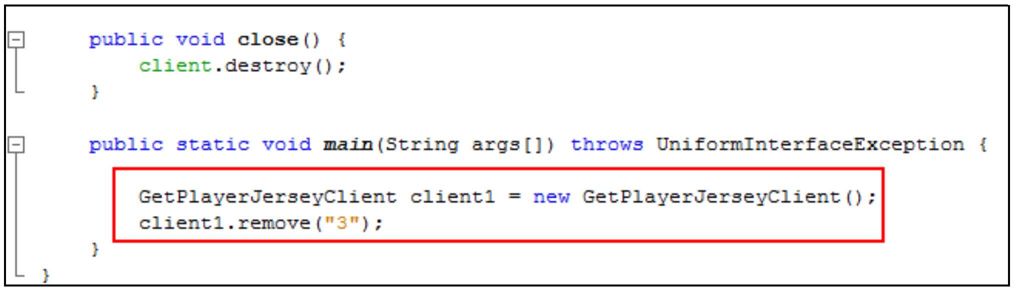
1 . To Delete a specified row from the PLAYER table using the RESTful Web Service, perform the following steps:

Make the following changes to main method in `GetPlayerJerseyClient.java`

a. Delete the contents of the `main` method.

b. Add the below lines of code to the `main` method :

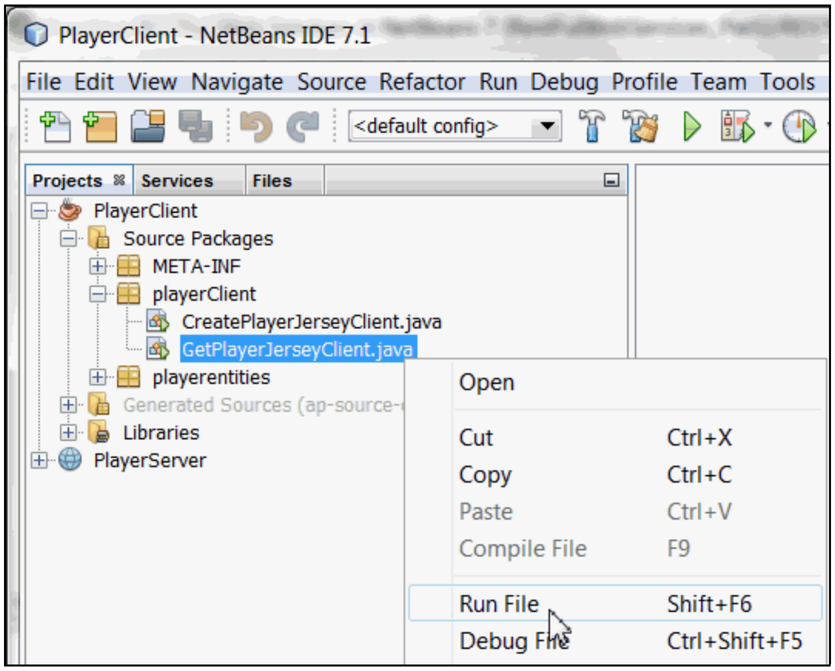
```
GetPlayerJerseyClient client1=new GetPlayerJerseyClient();
client1.remove("3");
```



```
    public void close() {
        client.destroy();
    }

    public static void main(String args[]) throws UniformInterfaceException {

        GetPlayerJerseyClient client1 = new GetPlayerJerseyClient();
        client1.remove("3");
    }
}
```

b. Examine that `remove()` method is used to delete a record from the Player table with PlayerId, 3.

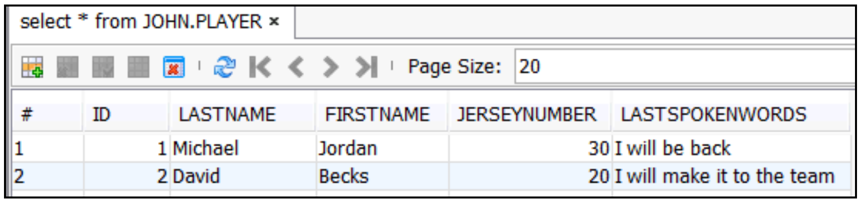
2 . In the Projects window, right-click `GetPlayerJerseyClient.java` and select **Run File** from the right-click menu.



3 . Verify the output. It can be done in two ways: examine the contents of the PLAYER table or Generate Web Services test client .

a. To examine the contents of the PLAYER table, complete the following steps:

1. In the **Services** window, expand the `jdbc:derby://localhost:1527/playerDB` connection under the Databases node.
2. Right-click the connection and select Refresh.
3. Expand the JOHN schema > Expand Tables Node >PLAYER Table.
4. Right-click PLAYER table node and select **View Data**.
5. The row with player id, 3 is deleted from 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

b. Generate Web Services test client as explained in Part-1. To test the Web Services Client, complete the following steps.

1. Select one resource node `com.playerentity.player`.
2. In the "Choose method to test" field, select `GET(application/xml)`.
3. Click Test.
4. Examine the output, player element with details of player id 3 is deleted.

Summary

This tutorial provides an overview of how RESTful WebServices and JPA together simplifies the implementation of database CRUD operations

Resources

JSR 311: JAX-RS: The Java API for RESTful Web Services
Jersey Implementation for building RESTful Web Services

Credits

Curriculum Developer: Anjana Shenoy