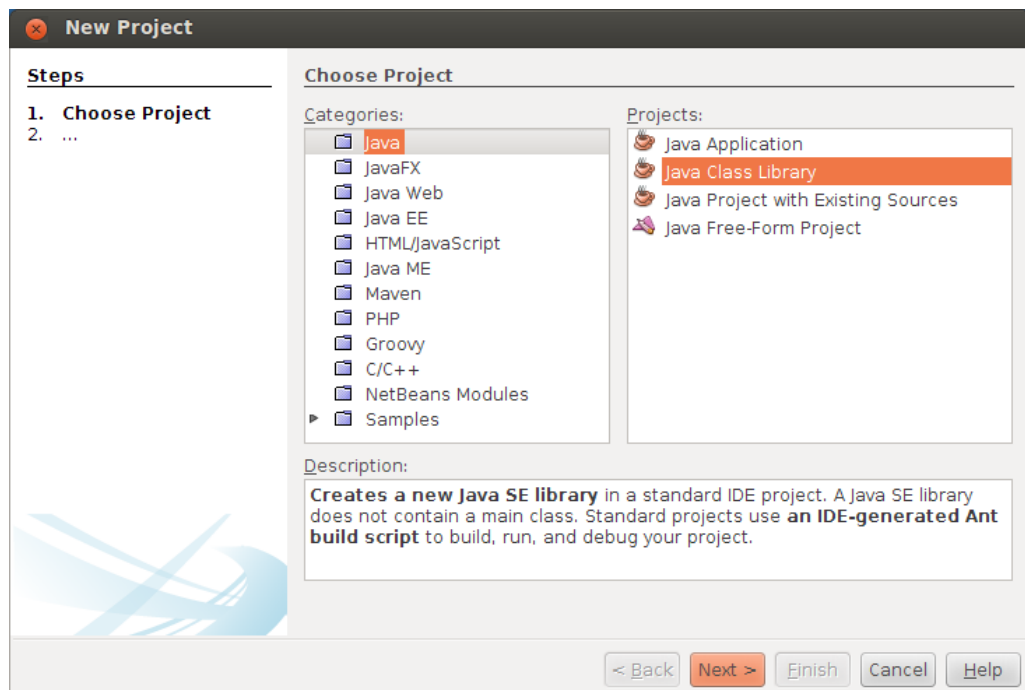


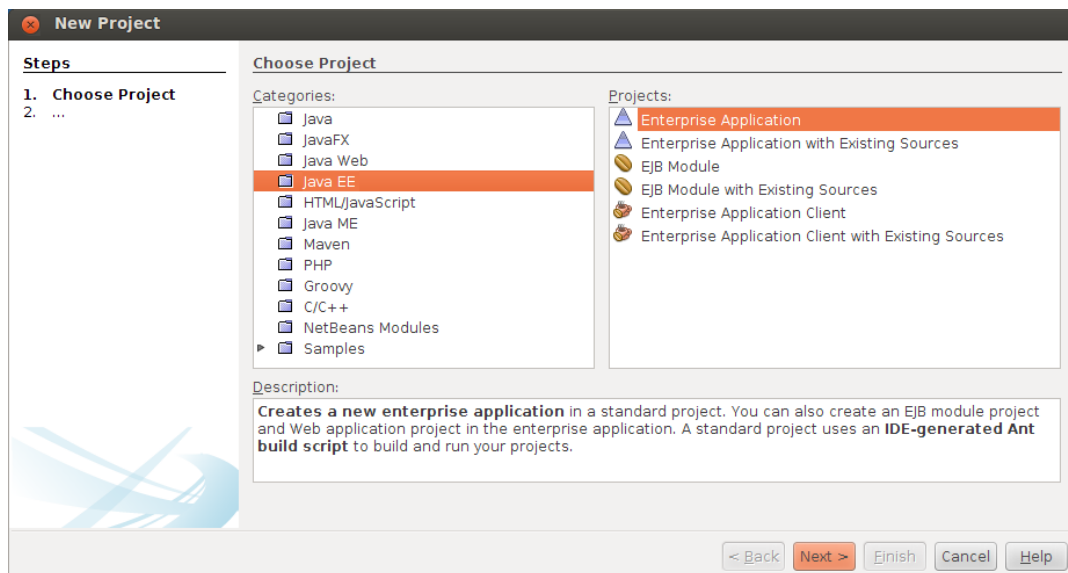
1. Create a new Java Class Library to hold the remote interfaces for your EJBs.

I created a new Java Class Library by going to File > New Project > Java > Java Class Library. For this project, I named it problemset3Lib.



2A. Create a new Java EE Enterprise Application containing an EJB Module.

To create a new Java EE Enterprise Application, I went to: File > New Project > Java EE > Enterprise Application. I named the project problemset3EnterpriseApp. On the last step of the gui wizard, I checked the box marked "Create EJB Module". This created an EJB module titled problemset3EnterpriseApp-ejb.



2B. Add a new Session Bean to your EJB Module. Name the bean HelloWorldBean. The EJB should be Stateless and should have a Remote interface.

To add a new session bean to my problemset3EnterpriseApp-ejb module, I right clicked on the module, and selected New > Session Bean. I named the bean “HelloWorldBean”, named it's package “ejb”, selected stateless from the session type, and checked the box “Remote” for the interface type. I then selected the class library I created in question 1, “problemset3Lib”, as the remote interface library.

New Session Bean

Steps

1. Choose File Type
2. Name and Location

Name and Location

EJB Name: HelloWorldBean

Project: problemset3EnterpriseApp-ejb

Location: Source Packages

Package: ejb

Session Type:

☒ Stateless

☐ Stateful

☐ Singleton

Create Interface:

☐ Local

☒ Remote in project: problemset3Lib

< Back Next > Finish Cancel Help

3. Add a Business Method String hello(String name) to your EJB that takes name as a parameter and returns “Hello, name.”

To create an EJB Business Method, I began by opening the file for my newly created session bean, HelloWorldBean.java. I right clicked an area of the editor inside of the HelloWorldBean class, and selected Insert Code > Add Business Method. From there, I named the method “hello”, with a return type String, with a remote interface, and clicked ok. The method definition was then added to HelloWorldBeanRemote.java, and the implemntation was added to HelloWorldBean.java.

Add Business Method...

Name: hello

Return Type: String Browse...

Parameters Exceptions

Name	Type	Final
name	java.lang.String	<input type="checkbox"/>

Add Remove Up Down

Use in Interface: ☐ Local ☒ Remote ☐ Both

OK Cancel

I then added the return line, to return Hello, + the parameter

```
package ejb;
import javax.ejb.Stateless;

@Stateless
public class HelloWorldBean implements HelloWorldBeanRemote {

    public String hello(String name) {
        return "Hello, " + name;
    }
}
```

4A. Create a new Enterprise Application Client and add it to your Enterprise Application.

To create a new Enterprise Application Client, I clicked File > New Project > Java EE > Enterprise Application Client, and named the project problemset3EnterpriseClient. On the “Server and Settings” page, I selected problemset3EnterpriseApp from the “Add to Enterprise Application” dropdown. Glassfish 4.0 was selected as the server, and Java EE 7 was chosen as the Java version.

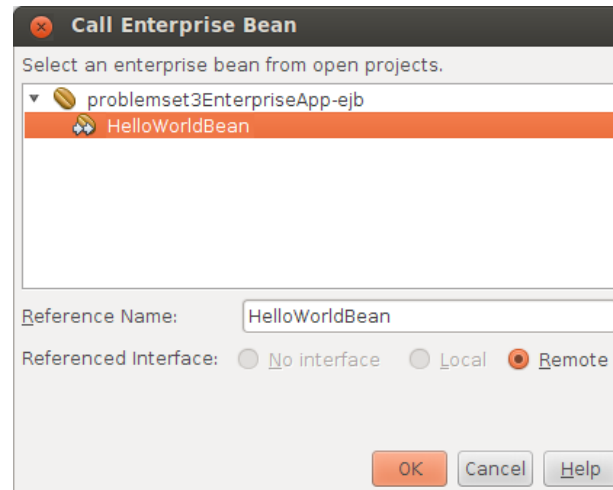
The screenshot shows the 'New Enterprise Application Client' dialog box with the 'Name and Location' tab selected. The 'Steps' list on the left indicates the current step is '2. Name and Location'. The 'Project Name' field contains 'problemset3EnterpriseClient'. The 'Project Location' field shows the path '/home/dan/school/CPSC476/problemset3'. The 'Project Folder' field shows the path '/home/dan/school/CPSC476/problemset3/problemset3EnterpriseClient'. There is an unchecked checkbox for 'Use Dedicated Folder for Storing Libraries' and a 'Libraries Folder' field with a 'Browse...' button. At the bottom, there are buttons for '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'.

Adding client to main project

The screenshot shows the 'New Enterprise Application Client' dialog box with the 'Server and Settings' tab selected. The 'Steps' list on the left indicates the current step is '3. Server and Settings'. The 'Add to Enterprise Application' dropdown menu is set to 'problemset3EnterpriseApp'. The 'Server' dropdown menu is set to 'GlassFish Server 4.0'. The 'Java EE Version' dropdown menu is set to 'Java EE 7'. The 'Main Class' field contains 'problemset3Enterpriseclient.Main'. At the bottom, there are buttons for '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'.

4B. In your app client's main() method, Call Enterprise Bean HelloWorldBean's hello() method and print its output.

I opened Main.java, under my enterprise client project. Inside the main class, I right clicked and selected Insert Code > Call Enterprise Bean. I selected HelloWorldBean from my problemset3EnterpriseApp-ejb. HelloWorldBean was the reference name, and referenced interface was set to remote. This created an instance of the bean. I then called it's hello() method with the name "Dan" passed to it. The result was output.



```
1 package problemset3EnterpriseClient;
2
3 import ejb.HelloWorldBeanRemote;
4 import javax.ejb.EJB;
5
6 public class Main {
7     @EJB
8     private static HelloWorldBeanRemote helloWorldBean;
9
10    public static void main(String[] args) {
11        System.out.println(helloWorldBean.hello("Dan"));
12    }
13 }
```

4C. Run your enterprise application and take a screenshot of the NetBeans Output window.

To run my enterprise application, I selected problemset3EnterpriseApp from my project listing. I right clicked, and selected deploy. After the Glassfish server started up, I clicked Run, and received the following output.

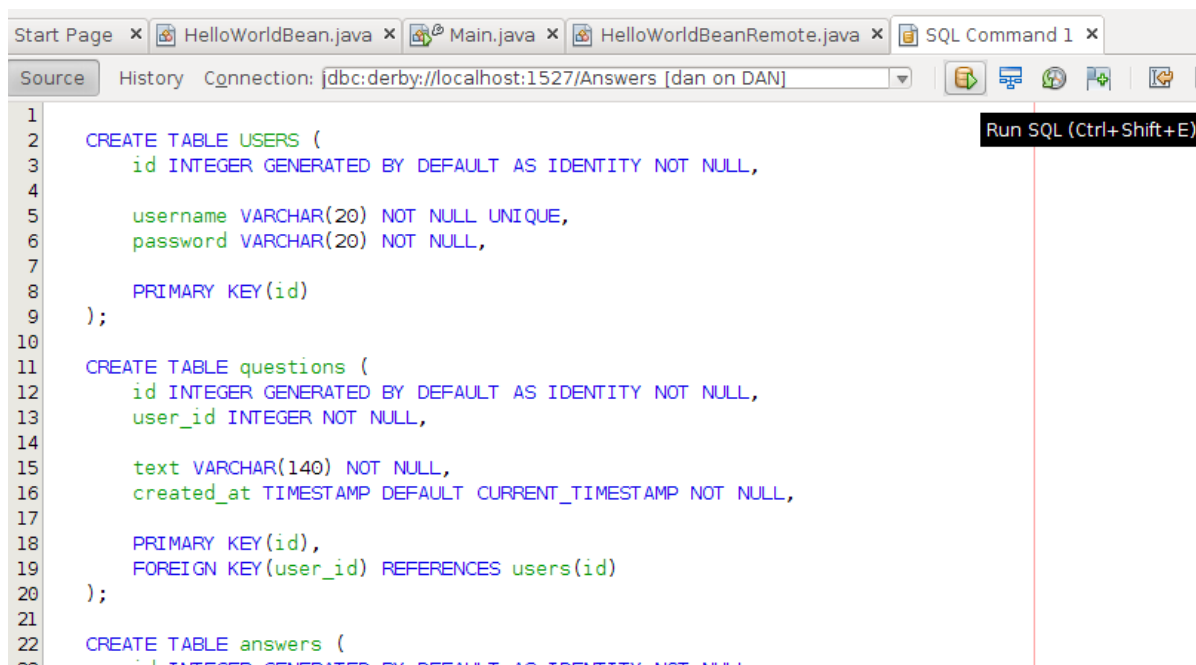
```
Initial deploying problemset3EnterpriseApp to /home/dan/school/CPSC476/problemset3/problemset3EnterpriseApp/dist/gfdeploy/problemset3EnterpriseApp
Completed initial distribution of problemset3EnterpriseApp
Initializing...
post-run-deploy:
run-deploy:
run-display-browser:
run-ac:
pre-init:
init-private:
init-userdir:
init-user:
init-project:
do-init:
post-init:
init-check:
init:
Copying 1 file to /home/dan/school/CPSC476/problemset3/problemset3EnterpriseApp/dist
Copying 3 files to /home/dan/school/CPSC476/problemset3/problemset3EnterpriseApp/dist/problemset3EnterpriseAppClient
Copying 6 files to /home/dan/school/CPSC476/problemset3/problemset3EnterpriseApp/dist/problemset3EnterpriseAppClient
Hello, Dan
run:
BUILD SUCCESSFUL (total time: 16 seconds)
```

6. Create a New Java DB database. Connect to the database, then Execute the Commands from the SQL files provided in Problem Set 1.

To create a new Java DB database, I selected Services, from the navigation window in the top left of the screen. Expanding the database listing, I right clicked on Java DB and selected Create Database. I created a database titled “Answers”, with the username “App” and password “password”



To connect to the database, I right clicked on jdbc:derby://localhost:1527/Answers [App on APP] and selected connect. To execute the SQL statements from problemset2, I right clicked the database, and selected “Execute Command”. This opened a new editor window, where I pasted in the SQL statements to create the Answers, Users, Questions tables. After pasting in the SQL, I clicked the “Run SQL” button in the top right corner of the editor.



After creating the tables, I verified that they were created successfully, by looking under APP > Tables in the services tab. I then executed the insert statements to populate the database.

```
Start Page x HelloWorldBean.java x Main.java x HelloWorldBeanRemote.java x SQL Command 1 x
Source History Connection: [dbc:derby://localhost:1527/Answers [App on APP]]
1 INSERT INTO users(id, username, password) VALUES(1, 'fishygut', 'password');
2 INSERT INTO users(id, username, password) VALUES(2, 'puffstone', 'qwerty');
3 INSERT INTO users(id, username, password) VALUES(3, 'barkingcustard', 'welcome');
4
5 INSERT INTO questions(id, user_id, text) VALUES(1, 1, 'What are the main differences between Java EE 7 and Java EE
6 INSERT INTO questions(id, user_id, text) VALUES(2, 3, 'Where can I find code examples for Java 7 EE Tutorial?');
7
8 INSERT INTO answers(user_id, question_id, text) VALUES(2, 1, 'Support for JSON');
9 INSERT INTO answers(user_id, question_id, text) VALUES(3, 1, 'GlassFish v4');
10 INSERT INTO answers(user_id, question_id, text) VALUES(2, 1, 'Improved Bean Validation');
11 INSERT INTO answers(user_id, question_id, text) VALUES(3, 1, 'WebSocket support');
12 INSERT INTO answers(user_id, question_id, text) VALUES(2, 2, 'https://java.net/projects/javaee/tutorial/sources/syn/
13 INSERT INTO answers(user_id, question_id, text) VALUES(3, 2, 'Thanks!');
14
```

To confirm that the data was successfully inserted, I ran a simple query, to return everything in the Answers, Users, Questions tables.

Source History Connection: [dbc:derby://localhost:1527/Answers [App on APP]]

```
1 SELECT * FROM APP.ANSWERS, APP.QUESTIONS, APP.USERS;
```

SELECT * FROM APP.ANSWERS... x

Page Size: 20 Total Rows: 36 Page: 1 of 2 Matching Rows:

#	ID	USER_ID	QUESTION_ID	TEXT	CREATED_AT	ID	USER_ID	TEXT
1	1	2	1	Support for JSON	2013-11-08 13:19:55.663	1	1	What are the main differences betwe...
2	2	3	1	GlassFish v4	2013-11-08 13:19:55.683	1	1	What are the main differences betwe...
3	3	2	1	Improved Bean V...	2013-11-08 13:19:55.718	1	1	What are the main differences betwe...
4	4	3	1	WebSocket supp...	2013-11-08 13:19:55.739	1	1	What are the main differences betwe...
5	5	2	2	https://java.net/...	2013-11-08 13:19:55.778	1	1	What are the main differences betwe...
6	6	3	2	Thanks!	2013-11-08 13:19:55.801	1	1	What are the main differences betwe...
7	1	2	1	Support for JSON	2013-11-08 13:19:55.663	1	1	What are the main differences betwe...
8	2	3	1	GlassFish v4	2013-11-08 13:19:55.683	1	1	What are the main differences betwe...
9	3	2	1	Improved Bean V...	2013-11-08 13:19:55.718	1	1	What are the main differences betwe...
10	4	3	1	WebSocket supp...	2013-11-08 13:19:55.739	1	1	What are the main differences betwe...
11	5	2	2	https://java.net/...	2013-11-08 13:19:55.778	1	1	What are the main differences betwe...
12	6	3	2	Thanks!	2013-11-08 13:19:55.801	1	1	What are the main differences betwe...
13	1	2	1	Support for JSON	2013-11-08 13:19:55.663	1	1	What are the main differences betwe...

6. In your Java Class Library, create New Entity Classes from the Database. Use the Database Connection for your new database, and Add All tables.

To create my entity classes, I right clicked on my java class library (problemset3Lib) and selected New > Entity Classes from Database. I selected ANSWERS, USERS, and QUESTIONS from the available tables, and clicked add all. On the next page, I made sure problemset3Lib was the listed projec, and ejb was the package name. I also checked the box generating named query annoations, and jaxb annoations.

The screenshot shows the 'New Entity Classes from Database' dialog box. On the left, a 'Steps' list indicates the current step is '3. Entity Classes'. The main area is titled 'Entity Classes' and contains a table for specifying class names and generation types. The table has three columns: 'Database Table', 'Class Name', and 'Generation Type'. It lists 'ANSWERS', 'QUESTIONS', and 'USERS' with corresponding class names and 'New' generation types. Below the table, the 'Project' is set to 'problemset3Lib', 'Location' is 'Source Packages', and 'Package' is 'ejb'. Three checkboxes are checked: 'Generate Named Query Annotations for Persistent Fields', 'Generate JAXB Annotations', and 'Create Persistence Unit'. At the bottom, there are buttons for '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'.

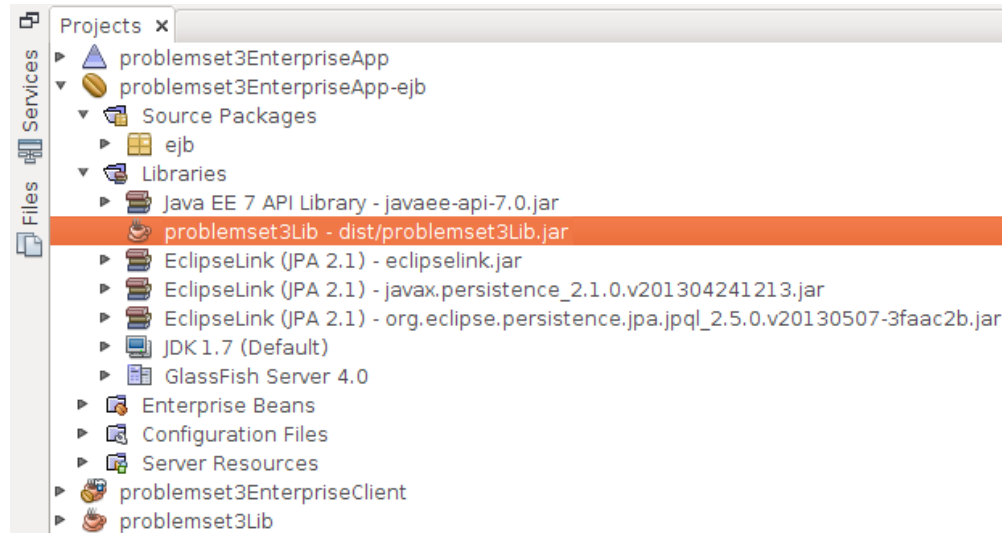
Database Table	Class Name	Generation Type
ANSWERS	Answers	New
QUESTIONS	Questions	New
USERS	Users	New

I also created an EclipseLink persistance unit, and created a datasource for my database

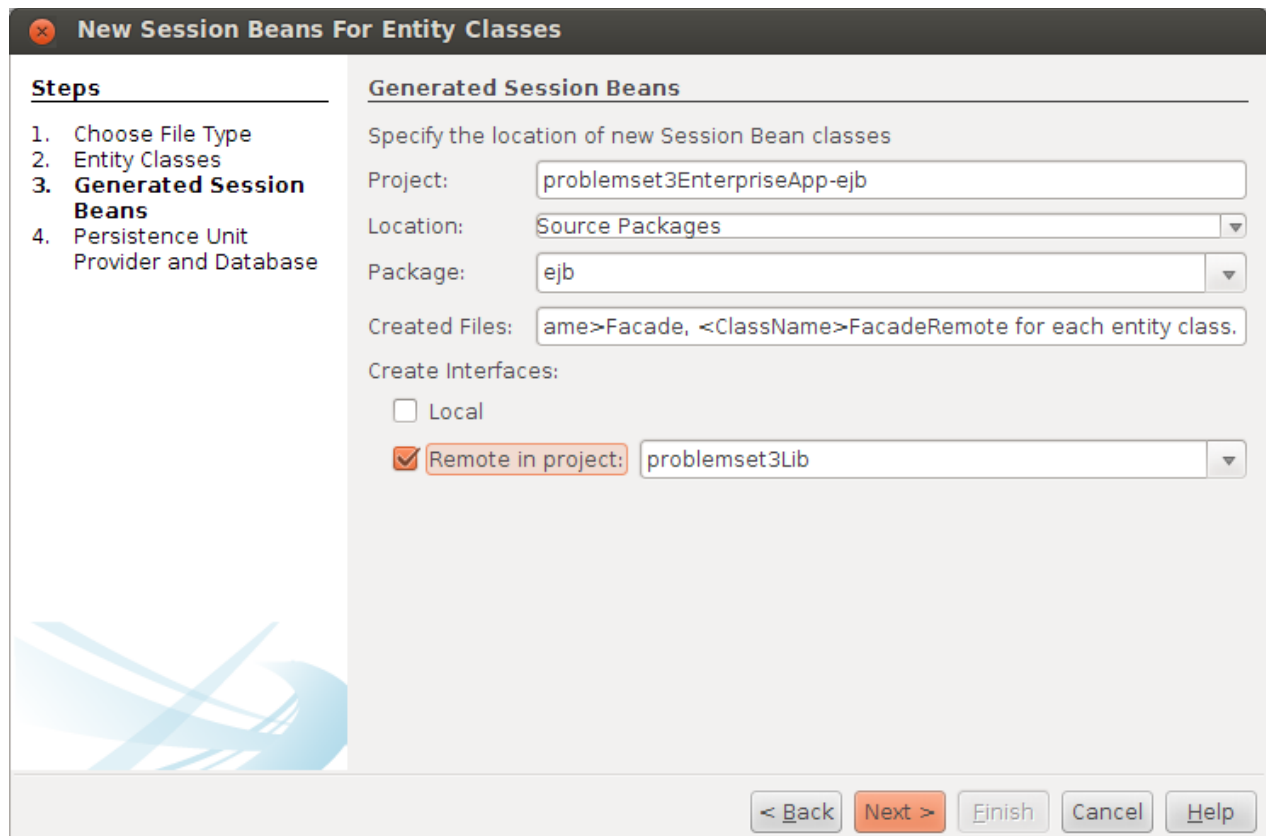
The screenshot shows the 'Create Data Source' dialog box. It has two main fields: 'JNDI Name' set to 'Answers' and 'Database Connection' set to 'jdbc:derby://localhost:1527/Answers [App on APP]'. At the bottom, there are buttons for 'OK', 'Cancel', and 'Help'.

7. Check that your Java Class Library is listed in the Libraries for your EJB Module, then create New Session Beans for Entity Classes. Your Generated Session Beans should have Remote interfaces in your Java Class Library project.

My Java class Library is listed in the libraries for my EJB Module, along with the Java EE libraries, and EclipseLink libraries

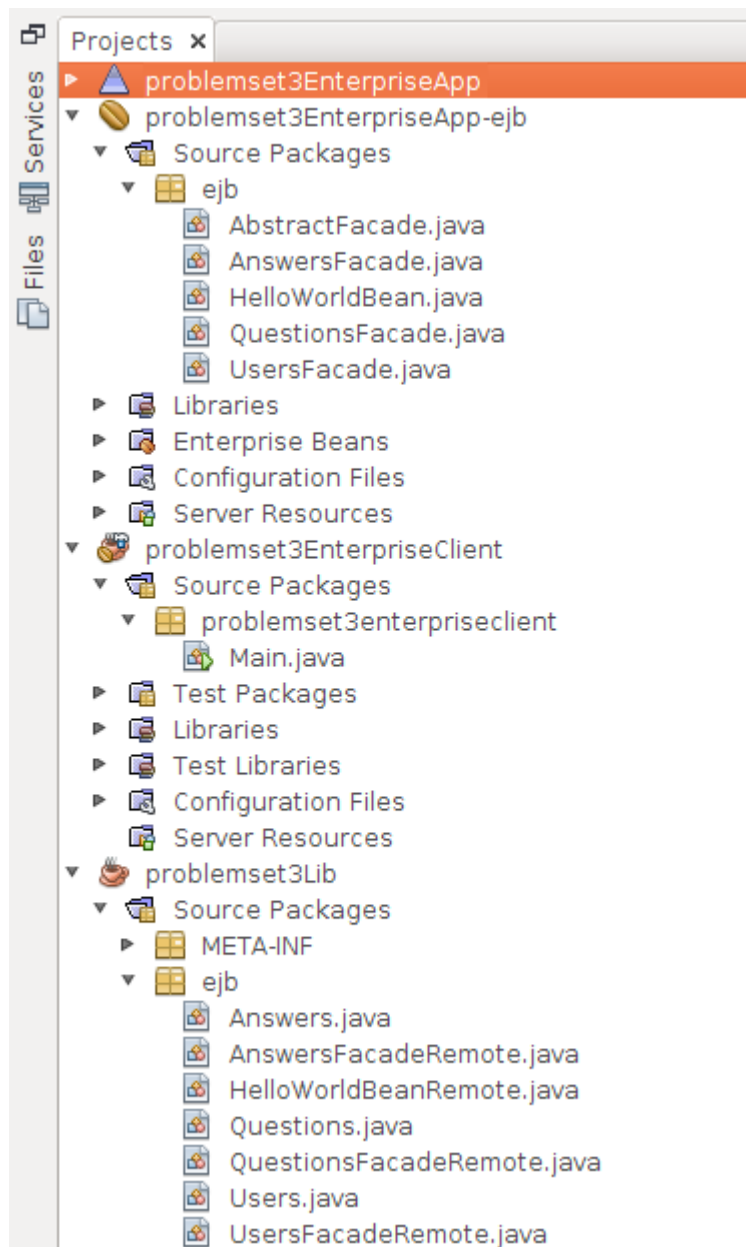


To create new session beans for my entity classes, I right click on my EJB module, and click New > Session Beans for Entity Classes. I select ejb.Answers, ejb.Questions, ejb.Users from the available entity classes, and click add all. On the next page, I check the box to create remote interfaces, and select my library, problemset3Lib.



This creates entity facade files in my ejb module, and remote interfaces in my problemset3Lib project.

At this point, this is what my project structure looks like



8. In your app client, Insert Code to call the new QuestionsFacadeRemote EJB. Use this EJB to generate the question report from Problem Set 2.

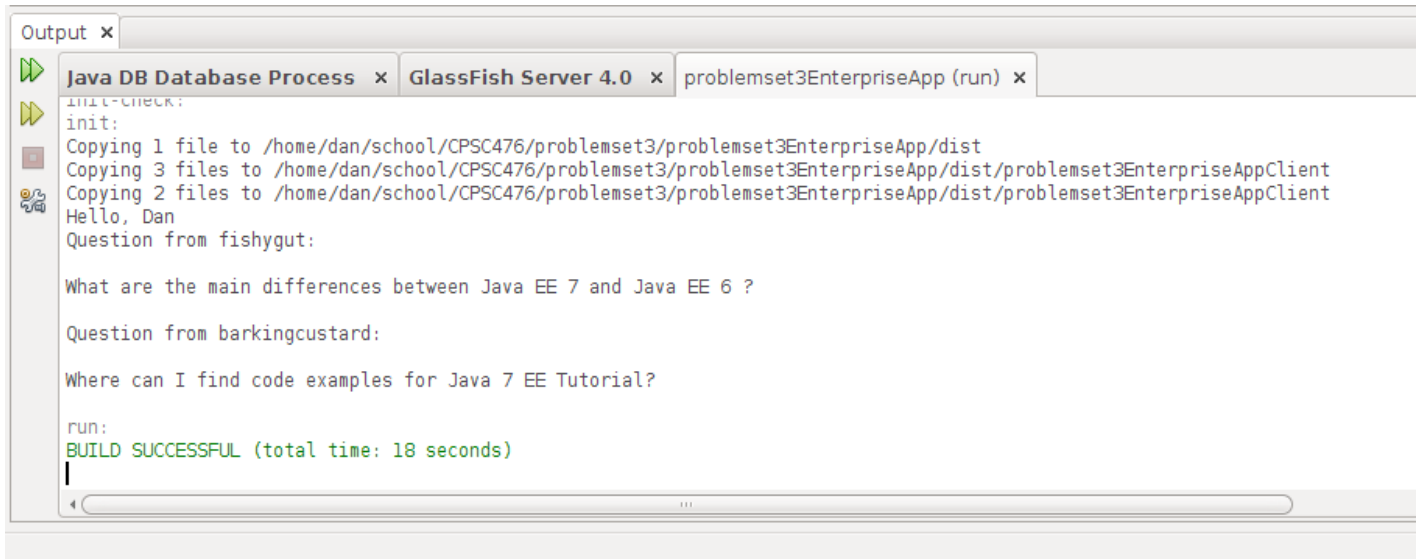
To get my EJB to connect to the database, I had to make a few changes to my persistence.xml file. First, I needed to change the persistence-unit name to problemset3EnterpriseApp-ejbPU, to match the name defined in my QuestionsFacade class. I also needed to change the transaction type of the persistence-unit from RESOURCE_LOCAL to JTA. Lastly, I needed to add a JTA data source, which was initially defined when creating my entity classes. The created datasource was called "Answers", so I added the line <jta-data-source>Answers</jta-data-source> just below the persistence-provider information.

I also had some issues relating to the ordering of libraries in my project. But by ordering the EclipseLink libraries ahead of the Java EE library, I was able to fix that problem and finally connect to my database.

I was able to retrieve the list of questions, and the usernames of each user asking the question, but I was unable to generate a list of answers for the questions. I attempted to retrieve the answers for each question by calling getAnswersCollection() on the Questions object created after calling findAll() on the questionsFacade EJB. In the netbeans generated entity files, a one-to-many relationship was created on the getAnswersCollection() method.

The relationship is mapped by questionId, so I assumed that while looping through the collection of answers, the following line would print out each individual answer: `System.out.println(answer.getQuestionId().getText());`. However, I was receiving a stacktrace mentioning uninstantiated LAZY relationships. Ultimately, I could not find a solution for retrieving the answers from a single questionsFacade EJB.

Here is the output of what I was able to complete



```
Output x
Java DB Database Process x GlassFish Server 4.0 x problemset3EnterpriseApp (run) x
init-check:
init:
Copying 1 file to /home/dan/school/CPSC476/problemset3/problemset3EnterpriseApp/dist
Copying 3 files to /home/dan/school/CPSC476/problemset3/problemset3EnterpriseApp/dist/problemset3EnterpriseAppClient
Copying 2 files to /home/dan/school/CPSC476/problemset3/problemset3EnterpriseApp/dist/problemset3EnterpriseAppClient
Hello, Dan
Question from fishygut:

What are the main differences between Java EE 7 and Java EE 6 ?

Question from barkingcustard:

Where can I find code examples for Java 7 EE Tutorial?

run:
BUILD SUCCESSFUL (total time: 18 seconds)
```