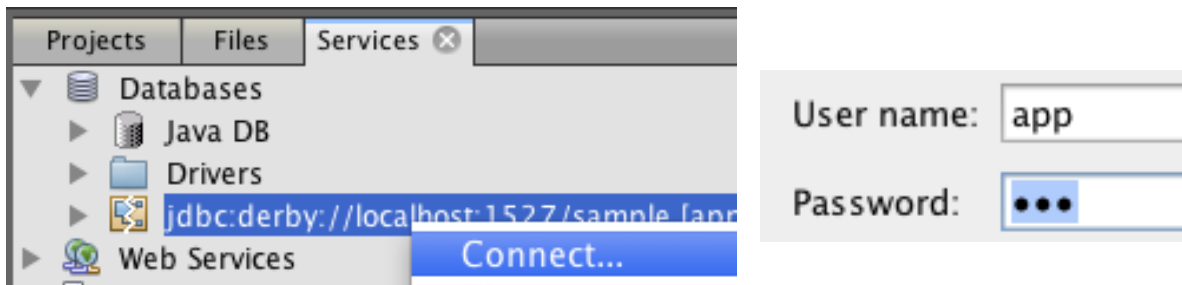


Lab 3 – Creating an application using JPA – Java Persistence API

Objective: Use JPA to run CRUD operations on the database table, with JSF for front end.

In this lab sheet we will be creating a new project, similar to lab sheet 2, but we will be storing the student registration details in a database table, and adding the functionality to maintain the table via JSF pages.

Netbeans ships with a Derby database, but has drivers to connect to many others include MySQL. We will be using the Derby database. To look at the database, select the Services tab, and right click on the jdbc:derby ... connection to connect to the database. The user name and password is **app**



Once connects, you can expand the jdbc:driver link to see the database schema.

Click on APP to open it, then click on Tables. You will see a number of sample tables. Note: the name of the connection string in SAMPLE, the persistence context will refer to it as **jdbc/sample**.

Right click on the PRODUCT table, and select View Data. This will open an SQL editor with the command **select * from app.product**. The SQL command has already been executed, and you can see the data from the table in the window below the SQL statement. You can run any SQL statement from this editor.

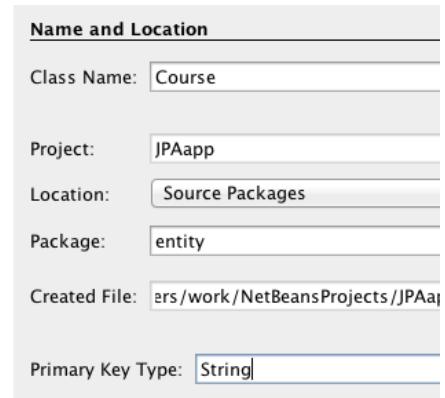
The rest of the lab sheet will talk you through creating a web app in Netbeans using JSF and JPA , and connects to the derby database.

Creating a JPA app with JSF

1. Return to the projects tab. As per lab sheet 2, create a new project. Choose Java Web, Web Application and click next. Call the project **JPAapp**, chose a project folder, and click next. On the server and settings window, make sure a **glassfish** server is selected, and click next. Under frameworks, click on **JavaServer Faces** and click finish.
2. We are going to create is an Entity class for course and for student, mapped to database tables. Right click on the Project name and select New > Entity Class (or select File > New file > Persistence > Entity Class).

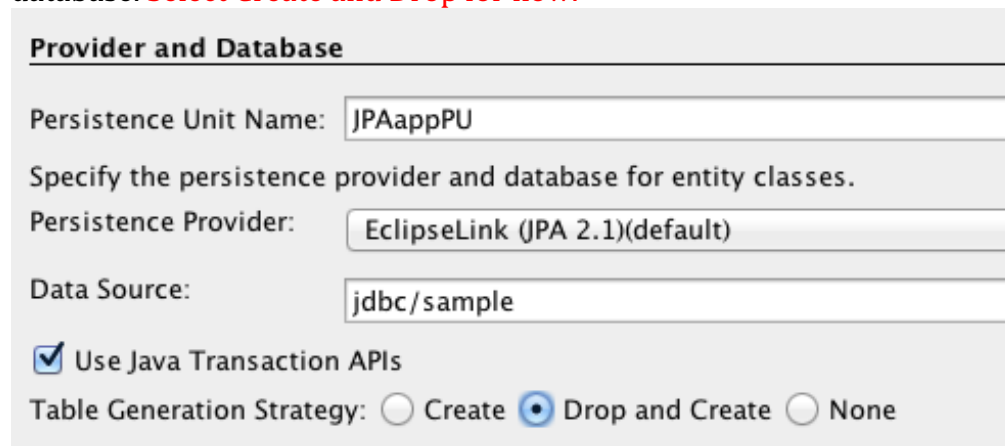
Lab 3 – Creating an application using JPA – Java Persistence API

3. The class name is **Course**, and the package is **entity**. Make the primary key a **String**. Make sure the 'Create Persistence Unit' is selected at the bottom, and click next.
4. Leave the persistent unit at the default name. Under data source, use the existing JNDI name for the database schema, i.e. **jdbc/sample**. (Note: if you wanted to use a different database, selecting New Data Source would allow you select an alternative database.)



Name and Location	
Class Name:	Course
Project:	JPAapp
Location:	Source Packages
Package:	entity
Created File:	ers/work/NetBeansProjects/JPAa...
Primary Key Type:	String

For the table generation strategy, if you select 'Create', a table called **course** will be created if it doesn't already exist. If you select 'Create and Drop', then every time you compile your project, the **Course** table SHOULD BE deleted from the database (it doesn't always work), and a new one created based on the attributes in the Entity class, so you would lose any data already in the database. **Select Create and Drop for now.**




Provider and Database	
Persistence Unit Name:	JPAappPU
Specify the persistence provider and database for entity classes.	
Persistence Provider:	EclipseLink (JPA 2.1)(default)
Data Source:	jdbc/sample
<input checked="" type="checkbox"/> Use Java Transaction APIs	
Table Generation Strategy:	<input type="radio"/> Create <input checked="" type="radio"/> Drop and Create <input type="radio"/> None

5. Click finish. Under **Source Package / Entites** you will now see is a new Entity Class called **Course.java**. **Double click on it to open it**. It is already defined as an entity class (**@Entity** annotation), and a primary key is defined (**@Id** annotation).
6. **Comment out the @GeneratesValue** annotation, as we don't want the primary key generated automatically. This will flag two unused imports which can also be commented out.
7. **Add two attributes to the table**, **courseDescription** and **department**, and generate the set and get methods:

```
private String courseDescription;  
private String department;
```

Lab 3 – Creating an application using JPA – Java Persistence API

8. Repeat step 3 to create a Student entity class. As per the backing bean we created last week, add the same attributes to this entity class, but giving course a type of Course:

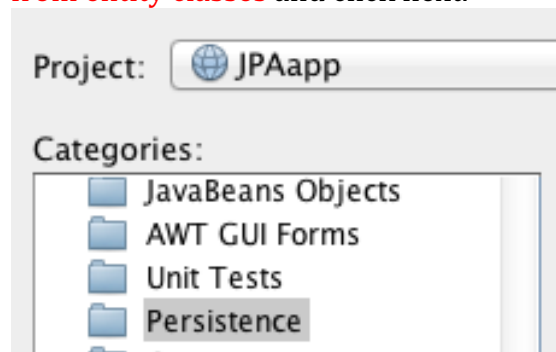
```
private Course course;  
private String firstName;  
private String lastName;  
private Integer age;  
private String email;
```
9. Under the attribute declarations, add `get` and `set` methods (**right click > select import code > select getter and setter methods**)
10. Defining course as a foreign key: You should see a warning icon beside the course attributes in the Student entity bean.  This is asking you if you want to use this attribute to link to the course table (i.e. define it as a foreign key in the underlying database table).

Click on the warning. You should be presented with four options regarding the type of foreign key to create. **Select bi-direction, manyToOne**. (Many students in one course). Next you will be asked what the corresponding 'link' attributes should be called in the course class. **Keep the default names of students**. This will add the following code to your entity beans:

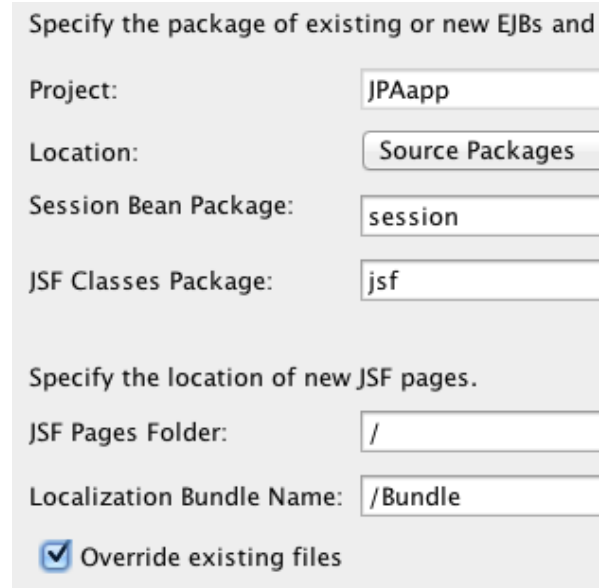
The course attributes in the student table is annotated as: `@ManyToOne`
The course table now has: `@OneToMany(mappedBy = "course")`
`private List<Student> students;`

11. Automatically generate the rest of the application:

Once the entity beans are correctly defined, Netbeans will generate the rest of the application for you. **Select file > new file > Persistence > JSF pages from entity classes** and click next.



Click on **entity.Student** and **entity.Course**, and select **Add** to bring them to the right hand side window. The right hand side window lists the



Lab 3 – Creating an application using JPA – Java Persistence API

entity classes for which JSF pages will be created. Click next.

This wizard generates quite a bit of code. To organise the code, use the package names shown on the right:

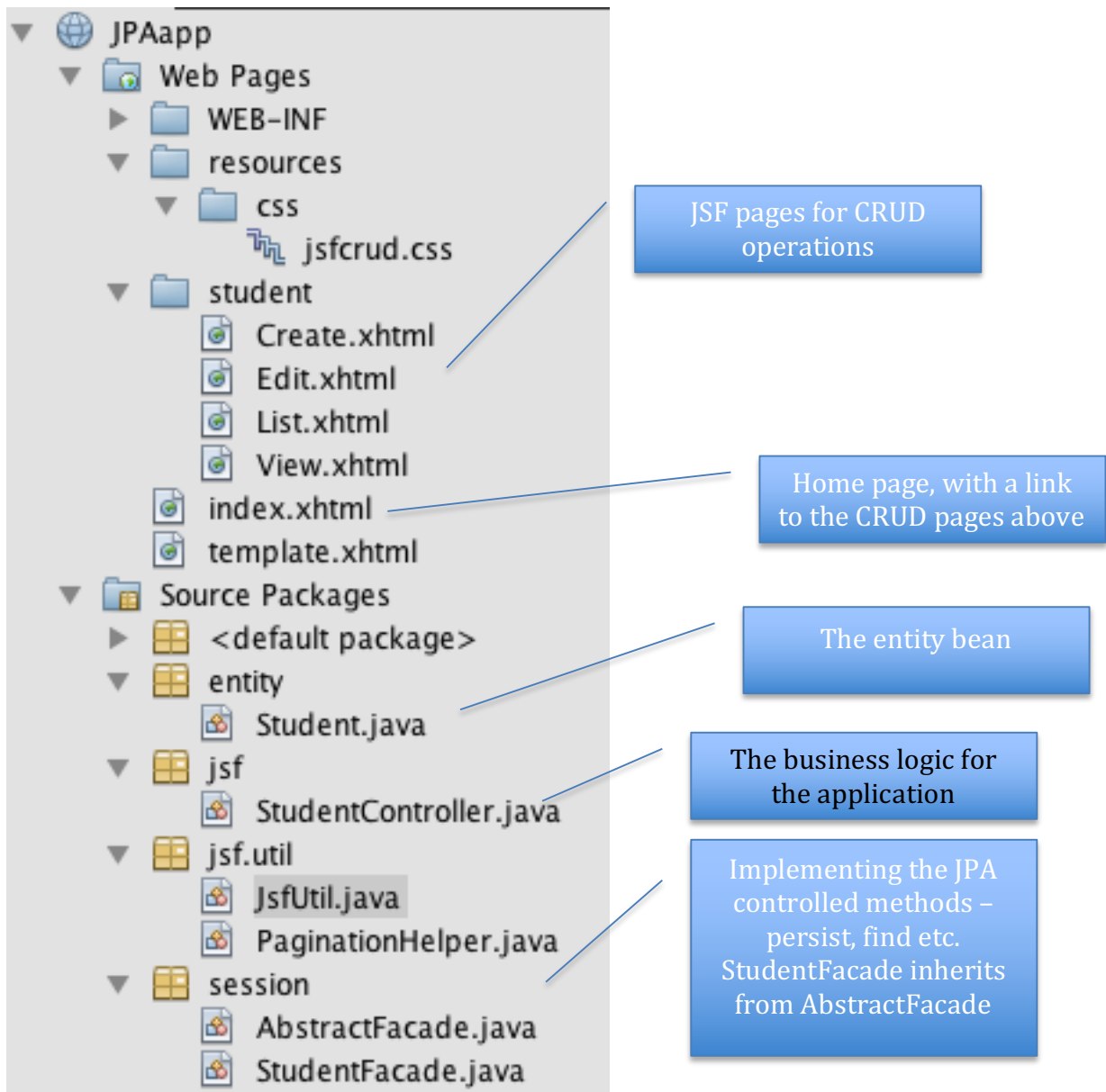
Session bean package: session

JSF classes package: jsf

JSF pages folder: /

Click finish.

12. **Take time to understand what has been generated.** Folders are explained below, with more details in the final two pages:



Lab 3 – Creating an application using JPA – Java Persistence API

13. Run the application. Add a course, and then add a student, selecting the course from the drop down box.
14. Return to netbeans, check that the new course and student have been added to the database tables.

Note: If attributes are missing from your XHTML pages, it could be due to missing get and set methods for those attributes.

15. Tidy up the front end, add form validation as per last week's lab, and improve the basic template.

Role of each file:

Bundles.properties: the text strings for labels in the HTML pages, allowing them to be changed here (e.g. for languages changes) rather than editing code to make changes.

.xhtml files: front end (web tier)

.java files: back end (business tier)

database: data tier

StudentController follows the MVC design pattern, and controls the application (i.e. what is called next etc.)

StudentFacade: this is where the JPA controller methods are implemented to add, delete, view and update data in the database. The methods are implemented in AbstractFacade and inherited by StudentFacade.

paginationHelper: manages page navigation for lists of students, if there are more students than will fit on one page.

Jsutil: additional functionality needed by the front end such as converting selects on a web page to a list of student objects, and converting parameters in the HTTP calls to objects.

Lab 3 – Creating an application using JPA – Java Persistence API

