# JDBC Project Description

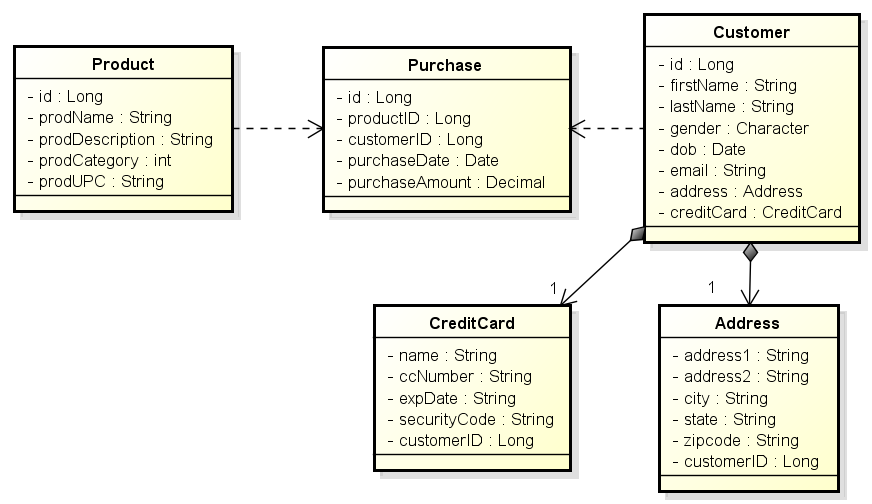
Project Due 4/4

## Introduction

This project is an introduction to developing services and DAOs (Data Access Objects) that support an application’s persistence requirements. The delivered services will persist a set of entity classes that are provided in the project materials.

# Entity Classes

The following classes represent the entities that will be persisted by the delivered services. Note that Java source files implementing these classes have been provided with the project materials. (JDBCProjectForStudent.zip)



One thing to note is that the Address and CreditCard entities do not provide an ID attribute. This is because they are defined as Weak Entities in the design. That is, Address and CreditCard do not have an identity of their own. Rather they are attached to Customer which is a Strong Entities in this design. This relationship can be illustrated in UML using the composite association i.e. with the filled black diamond. Customer is a Strong Entity in that it maintains a unique identity, and that makes it selectable, etc. Likewise, Purchase is also Strong Entity even though it maintains an association with Customer. Product is also a Strong Entity.

# Service & Interfaces

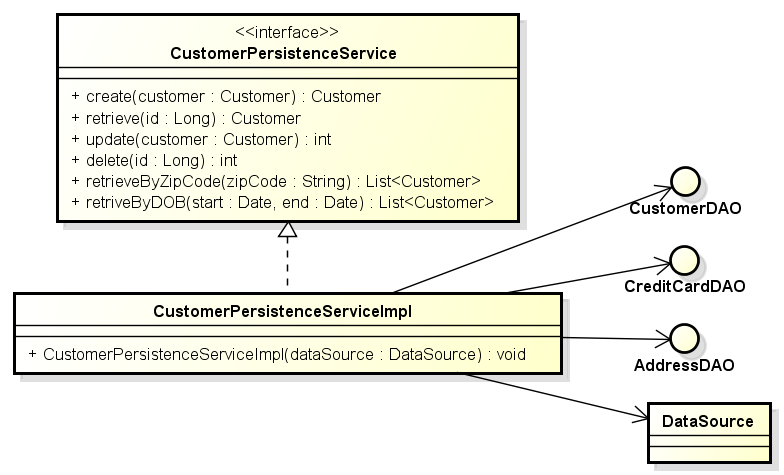
The following illustrates the Service interfaces that will be delivered with the project. Each of these services maintains a DataSource that is used to build the JDBC Connection instances that are passed to the DAO methods to CRUD and query for instances of the associated entities.

The delivered classes must implement these interfaces for the grading / testing to execute correctly. Note that Java interfaces have been provided with the project materials.

## CustomerPersistenceService

The CustomerPersistenceService is responsible for managing the persistence of Customer, Address, and CreditCard entities.

This service is to be implemented so that every operation on a Customer also manipulates its associated Address and CreditCard instances. For example, retrieving a Customer though its ID, also returns the Customer’s Address and CreditCard. Deleting a Customer also removes its Address and CreditCard. Etc.

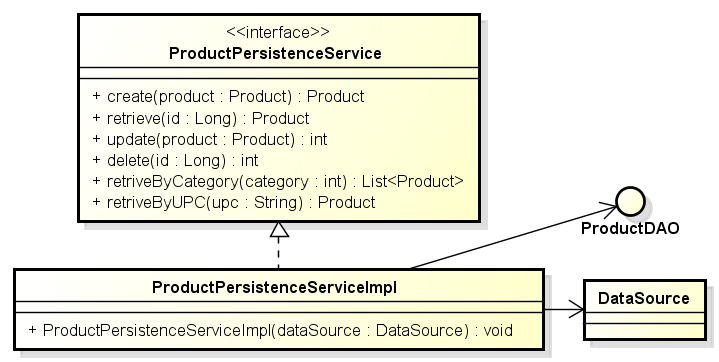


The CustomerPersistenceService provides the standard CRUD operation that allows the persistence and management of these instances. The service also provides operations that query for collections of Customer (Address & CreditCard) based on search criteria zip-code and date of birth ranges.

Notice that Address and CreditCard have not been provided persistence services in this design. This is because these are weak entities and attached to their Customer strong entity objects and all operations on a Customer are applied to their dependent Address & CreditCard objects.

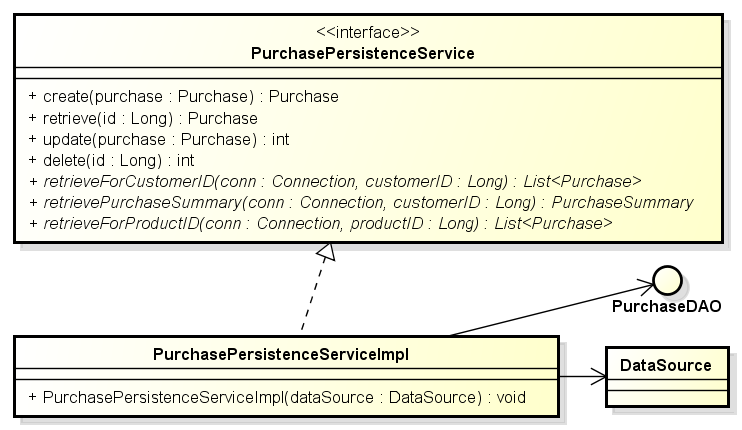
## ProductPersistenceService

The ProductPersistenceService provides the standard CRUD methods for Products maintained in the ecommerce sites inventory. The service also provides operations that allow for the query for Products in certain product categories and retrieving Products by their UPC (Universal Product Code).



## PurchasePersistenceService

The PurchasePersistenceService provides the standard CRUD methods for Purchases made by Customers for Products. The services also allows for the retrieval of Purchases made by specific Customer, and for the retrieval of Purchases for specific Products. The service also provides a purchase summary of purchases made by specific customers.

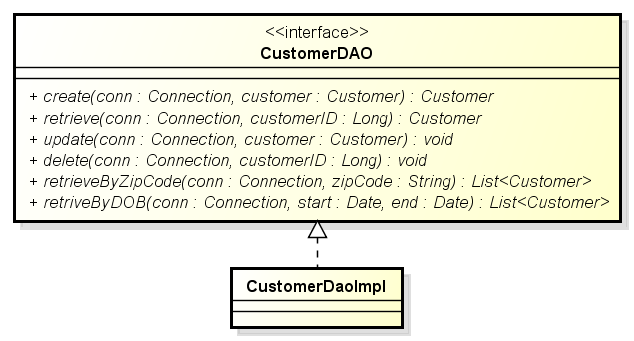


# DAO Interfaces

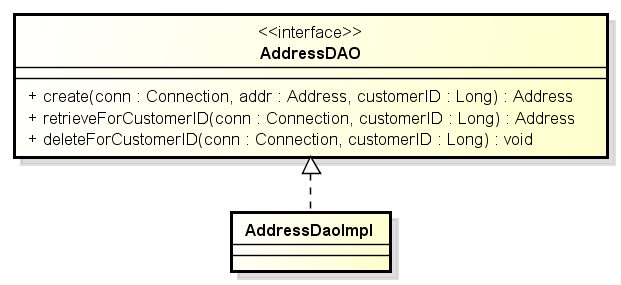
The following illustrates the DAO interfaces that will be delivered with the project. The delivered classes must implement these interfaces for the grading / testing to execute correctly. These Java interfaces have been provided with the project materials.

Notice that each of the DAO methods requires an open JDBC Connection instance that the operation uses to interact with the DBMS. The reason for passing the Connection to each method is that it allows the service classes that utilize the DAOs to manage the connection and to create transactions that span several DAO / JDBC / SQL operations.

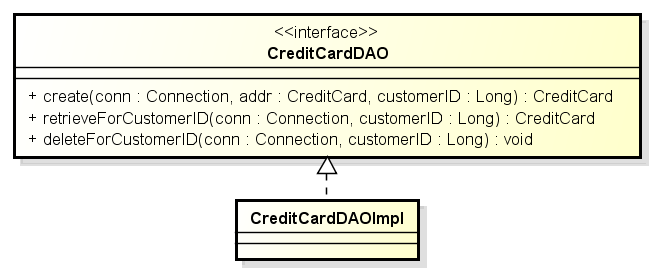
## CustomerDAO



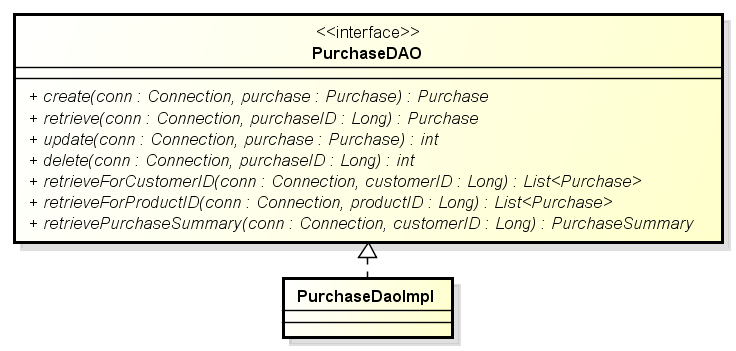
## AddressDAO



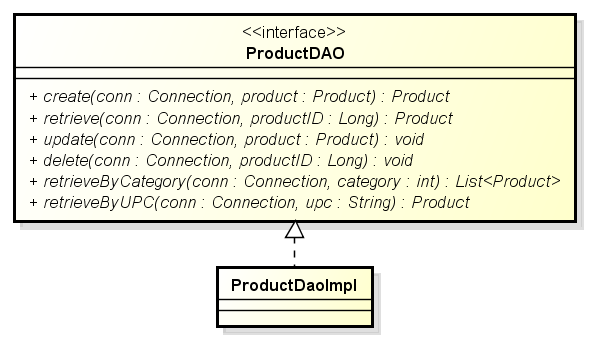
## CreditCardDAO



## PurchaseDAO



## ProductDAO



# Project Goals

The following are the goals of this project:

* To build a database schema that represents the given entity classes.
* To deliver an SQL script that will create the ‘simple\_customer’ schema on the instructor’s PC for grading purposes. This can be accomplished by first creating the schema in an ER diagram and then generating the schema’s SQL DDL using MySQL workbench.
* To deliver a set of Service and DAO implementation that successfully compile against, and can pass the Unit Tests that have been provided in the JDBCProjectTesting project provided in the project’s materials.
* To implement and deliver a set of five DAO classes based on provided interfaces that will be used to CRUD information against the delivered schema. These implementations must pass the Unit Tests provided in the project materials.
* To populate the schema tables with sample data using the application **PopulateTables.java** which has been provided in the testing project materials.
* To implement and deliver a Customer persistence service based on a provided CustomerPersistenceService interface that will be used to CRUD Customer entities against the delivered schema. This implementation must pass the Unit Test provided in the testing project materials.
* To implement and deliver a Product persistence service based on a provided ProductPersistenceService interface that will be used to CRUD Purchase entities against the delivered schema. This implementation must pass the Unit Test provided in the testing project materials.
* To implement and deliver a Purchase persistence service based on a provided PurchasePersistenceService interface that will be used to CRUD Purchase entities against the delivered schema. This implementation must pass the Unit Test provided in the testing project materials.
* **To deliver a library jar file that will be used to grade the project**. This jar will contain the contents of the JDBCProjectForStudents project and will be executed on the instructor’s PC and database to evaluate the quality of the team’s work. See the section “Exporting an Eclipse Project as a Library JAR File” at the end of this document.

# Provided Materials

You have been provided two archived Eclipse projects (zip files) that will be imported into your Eclipse workspace. The last section of this document provides instruction on how the import is accomplished.

These projects are:

## JDBCProjectForStudents

This project provides the basis for your DAO and Services implementations. This includes the Entity classes, DAO and Service interfaces as Java source. Your implementation will be written within this project, using these classes / definitions. The library jar file you create and submit for grading will be built from the work performed in this project.

**Do not change either the provided interfaces or the packaging structure. If changes are made, the unit tests will not compile (much less execute) and your team receives no credit for the work.**

## JDBCProjectTesting

This project contains programs and unit tests that you will use to configure and test your DAO and service implementations.

**Teams must not modify the code / contents of this project (With one exception. See section Notes on Unit Test)**. Changing the contents of the testing project will likely result in the unit tests in the instructor’s own unmodified testing project no longer working (no passing) and the loss of project points.

The JDBCProjectTesting project includes:

1. The application PopulateTables.java uses built-in JDBC to populate the database generated by your team’s schema. This includes a number of CSV files containing the data used to populate the CUSTOMER, ADDRESS, CREDIT\_CARD, etc. tables.
2. A number of Junit unit test classes which will test the correctness of your DAO implementations.
3. A number of Junit test classes that will test the correctness of your CustomerPersistenceService, PurchasePersistenceService, and ProductPersistenceService implementations.
4. The class DataSourceManager.java that is used to initialize (configure) and provide access to a singleton instance of a DataSource used by the unit test classes. **Note**: The text file dbconfiguration.properties contains the DataSource configuration and must be updated with the parameters needed to access your MySQL installation. You must not modify the URL’s schema (simple\_company).
5. A sample Java class SampeCreateMethod that is intended to provide an example of extracting the auto-increment primary key assigned to entity classes that are created (inserted).

# Suggested Development Process

The following steps detail a suggested process for executing this project.

1. Import both projects (Student and Testing) into your Eclipse workspace. **Note** that initially the project JDBCProjectForStudent will contain compile errors as the DAO and service implementations have not been completed (that is for you to do).
2. Be sure that the Testing project’s Eclipse build path includes the Student project.
3. You can use the main() application embedded in the DataSourceManager to verify that your application dbconfig.properties is correctly configured to connect to your running MySQL server.
4. Define a schema that meets the needs of the given entity classes. It is suggested that you use MySQL Workbench and that you export the CREATE SCHEMA SQL you will need to submit from workbench. The schema’s name must be ‘simple\_company’.
5. Implement some of the DAO interfaces against your schema. That is, implement the DAO with queries that are being utilized by the PopulateTable.java application. You need those services in place to populate the tables with test data in the next step (6).
6. Use the program PopulateTables.java (provided in the Test project) to populate the database from the data contained in the CSV files. Note that executing this application requires completing the insert\* methods in the DAO Implementation classes in the student development project.
7. Implement the remaining DAO interfaces against your schema. That is, complete the DAO implementation classes with queries that reflect your schema.
8. Use the provided DAO unit tests (see the package \*.unitTesting.dao) to validate the ‘correctness’ of your DAO implementations.
9. Implement the CustomerPersistenceService interface against your DAOs.
10. Implement the ProductPersistenceService interface against your DAOs.
11. Implement the PurchasePersistenceService interface against your DAOs.
12. Use the provided service unit test (see the package \*.unitTesting.service) to validate the correctness of your service implementations.
13. When all the tests work correctly, or you run out of time, package ONLY the contents of the JDBCProjectForStudents project into a library jar file that will be submitted for grading. See the section “Exporting an Eclipse Project as a Library JAR File” at the end of this document.
14. Test that the library jar file works correctly by including the library jar in the JDBCProjectTesting project’s build path. NOTE: You will need to remove the project JDBCProjectForStudents from the testing project’s build path for the validation to work.

# Notes on Unit Tests

Some of the unit tests rely on there being rows with specific IDs. For example “AddressDAOTest.java” has the constant customerID which defaults to 7. This test will fail a Customer in with this ID is not found in the database.

This customer will be created when the schema is re-installed and the populate application is run. This is because the auto-increment will start over from zero.

You can also change the value of the constant to an ID that is found in the corresponding table. This is the only change to the unit test source code that is allowed.

# Deliverables

Each team will deliver their project on a USB thumb drive that contains the following material. Note that the drive will be returned.

* A readme file (text or Word) that includes the team number, and all contributing team member names and their Net-IDs.
* An execution ready SQL script that generates the simple\_company schema when executed from MySQL workbench on the instructor’s PC.
* A library jar file exported from your JDBCProjectForStudents project. This jar used by the instructor’s JDBCProjectTesting Eclipse project to executes the unit tests.
* The project’s source code. You can deliver the source code by copying only the src directory or the entire JDBCProjectForStudents project directory onto the USB thumb drive.

# Grading Criteria

1. **20 Points**. Providing a library JAR file from JDBCProjectForStudents that correctly compiles against the Eclipse project JDBCProjectTesting on the instructor’s personal Eclipse workspace. This will be accomplished by importing the submitted jar into the JDBCProjectTesting project build path.
2. **20 Points**. Providing a SQL script that when executed on MySQL Workbench, creates a schema named ‘simple\_company’ (tables, etc.) on the instructor’s MySQL installation. (The schema name is important because it is reflected in the JDBC URL used to setup the DataSource for testing). The schema must reflect the structure of the entity classes provided.
3. **10 Points**. Executing the application PopulateTable.java and successfully populating the Customer, Address, CreditCard, Product, and Purchase tables created by your team’s schema in step 2.
4. **40 Points**. Executing each of the nine DAO and Service unit tests provided in the JDBCProjectTesting packages \*.unitTesting.dao and \*.unitTesting.service packages without error. Roughly 5 Points per Test.
5. **10 Points** is reserved for an evaluation of the quality of the design and code style.

# Implementation Requirements

Teams must not modify the code / contents of the Eclipse project JDBCProjectTesting. Doing so will likely result in the loss of points when the unit tests in the instructor’s own unmodified project no longer work.

The application DataSource will be configured with the contents of the property file ‘dbconfig.properties’ in the JDBCProjectTesting project. A property file has been provided with attaches to the DBMS running on the local machine. You will need to modify the id (maybe) and password. Do not change the default schema name (simple\_company).

The CustomerDAO, ProductDAO, and PurchaseDAO create() methods must throw a DAOException if the given Customer, Purchase, and Product has a non-null ID attribute.

The CustomerDAO, ProductDAO, and PurchaseDAO create() methods must return the given entities with the ID attribute assigned the key value provided by the auto-increment primary key column. An example of how this is accomplished in code has been provided.

The DAO retrieve\*() methods must return null if retrieving a non-existent ID.

# Using Eclipse

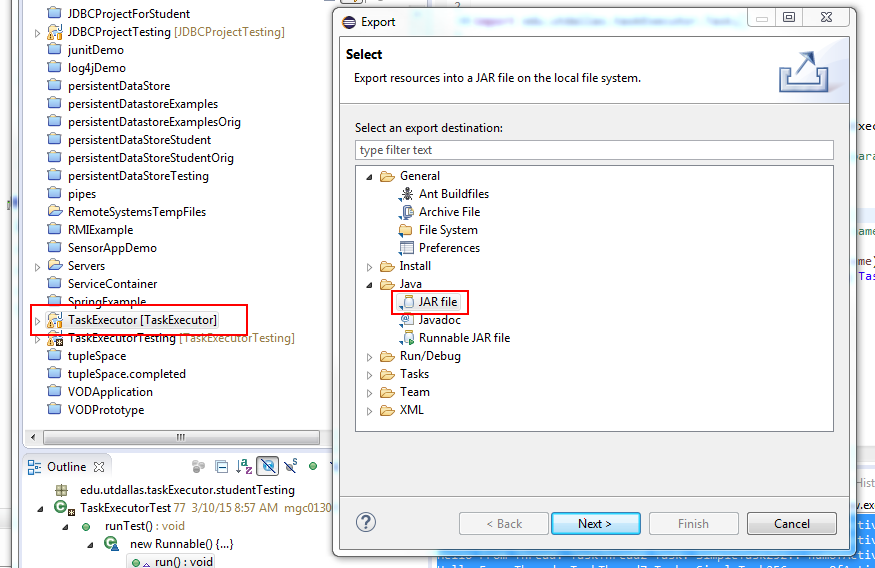
## Executing Unit Tests in Eclipse

1. You may need to install the JUnit library on the testing project.
2. You can select individual JUnit test cases (Test Classes), right click, and select “Run As JUnit Test”. This can also be done on a package to run all the tests in the package or on the project to run all test cases in the project.

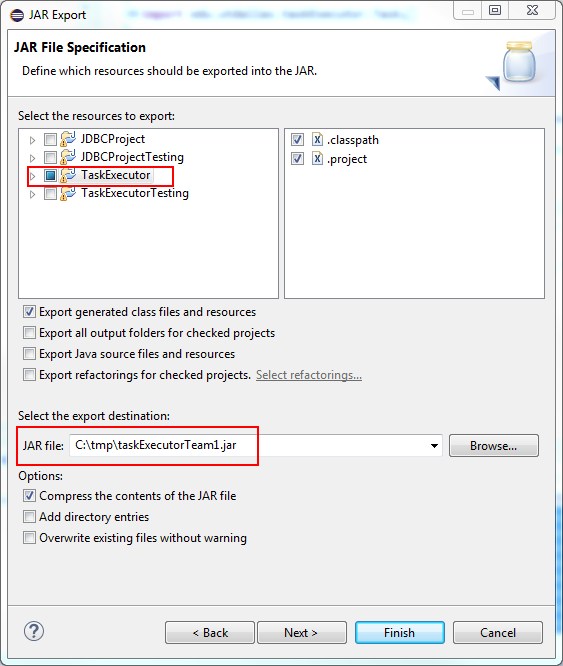
## Exporting an Eclipse Project as a Library JAR File

This section provides a procedure describing of how to export the project containing your team’s TaskExecutor implementation as a library .jar file for submission.

1. Select the project that you wish to export.
2. Use the right mouse button, or the file menu, to select the Export feature.
3. Select Java >>JAR File as shown below, and then Next.



1. On the JAR Export panel, make sure that the desired project is selected and enter the path and file name for the exported library jar file.
2. Select Finish and the export operation will be completed.

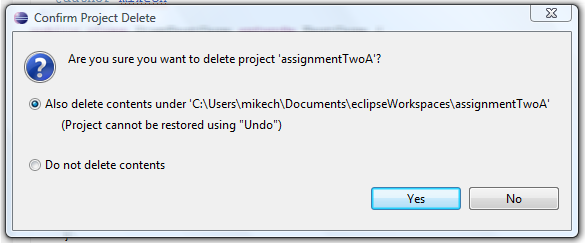


## Importing a Project into Eclipse from a Project Zip File

This programming assignment provides two exported projects that must be imported into your Eclipse workspace. These projects are provided as zip file archives that will be one of the files that can be downloaded from the WebCT assignment. The zip archive may contain sample code or a project template that can be used as a starting point for your efforts. You will be importing the project zip archive into your workspace.

**Optional: Removing existing projects with the same name from the workspace**

You cannot import a project with the same name into the workspace. This means that if you import and try to re-import the project template you must first delete the old project from the workspace. This is accomplished by selecting the existing project from the package explorer and selecting the “Edit > Delete” menu item. This will bring up the dialog shown in the following graphic. Notice that the option “Also Delete Contents Under C:\...” is selected**. It is very important that this option is selected** so that the project files are removed from you workspace

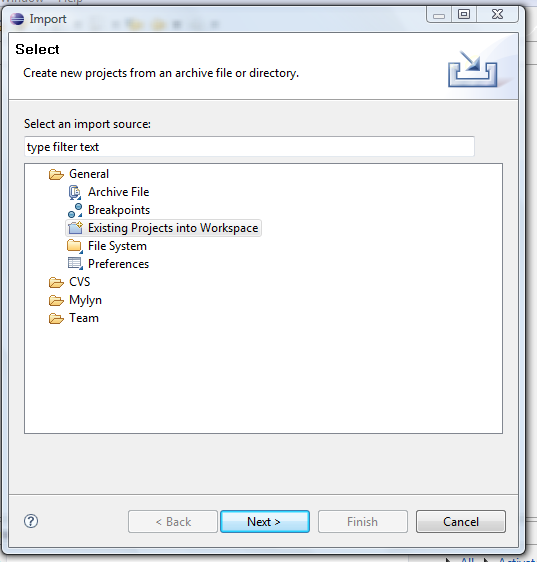


A this point the Old project will be have been removed from your workspace and you may begin importing the project template

## Importing the Project

The process for importing the template project is a follows.

Open the import wizard using the “File > Import” menu item. This brings up the import dialog shown in the following graphic. Make sure to select the “Existing Projects into Workspace” option (under General) and press Next.



This brings up the following import dialog. There are a few import things to note:

1. You need to select the “Select archive file” option and then press browse to select the project template archive (zip) file.
2. When the file opens, you need to select the project.
3. Press Finish and the project will be imported into your workspace.

