|  |  |
| --- | --- |
|  | **Cognizant Academy**  **Courier Tracking System**  **Java Specification Document**  **Version 1.0** |
| |  |  |  |  | | --- | --- | --- | --- | |  | **Prepared By / Last Updated By** | **Reviewed By** | **Approved By** | | **Name** |  |  |  | | **Role** |  |  |  | | **Signature** |  |  |  | | **Date** |  |  |  | |
|  |

Table of Contents

[1.0 Introduction 4](#_Toc21638735)

[1.1 Purpose of this document 4](#_Toc21638736)

[1.2 Definitions & Acronyms 4](#_Toc21638737)

[1.3 Project Overview 4](#_Toc21638738)

[1.4 Scope 4](#_Toc21638739)

[1.5 Intended Audience 4](#_Toc21638740)

[1.6 Hardware and Software Requirement 4](#_Toc21638741)

[1.7 Eclipse Project Configuration 5](#_Toc21638742)

[2.0 Class Diagram 5](#_Toc21638743)

[2.1 VO package 5](#_Toc21638744)

[2.2 Util Package 7](#_Toc21638745)

[2.2.1 Db.java 7](#_Toc21638746)

[2.3 DAO package 8](#_Toc21638747)

[3.0 Design for Validation of Admin 9](#_Toc21638748)

[3.1 Class Diagram 9](#_Toc21638749)

[3.2 AdminStaffDAO.java 10](#_Toc21638750)

[3.3 AdminStaffDAOImple.java 10](#_Toc21638751)

[4.0 Design for Validation of Staff 11](#_Toc21638753)

[4.1 Class Diagram 11](#_Toc21638754)

[4.2 AdminStaffDAO.java 12](#_Toc21638755)

[4.3 AdminStaffDAOImple.java 12](#_Toc21638756)

[5.0 Design for Registration of Admin and Staff 13](#_Toc21638758)

[5.1 Class Diagram 13](#_Toc21638759)

[5.2 AdminStaffDAO.java 14](#_Toc21638760)

[5.3 AdminStaffDAOImple.java 14](#_Toc21638761)

[6.0 Design for Registration of User 15](#_Toc21638763)

[6.1 Class Diagram 15](#_Toc21638764)

[6.2 UserDAO.java 16](#_Toc21638765)

[6.3 UserDAOImple.java 16](#_Toc21638766)

[7.0 Design for Validation of User 17](#_Toc21638767)

[7.1 Class Diagram 17](#_Toc21638768)

[7.2 UserDAO.java 18](#_Toc21638769)

[7.4 UserDAOImple.java 18](#_Toc21638771)

[8.0 Design for Retreiving the package details of user 19](#_Toc21638773)

[8.1 Class Diagram 19](#_Toc21638774)

[8.2 UserDAO.java 20](#_Toc21638775)

[8.3 UserDAOImple.java 20](#_Toc21638776)

[9.0 Design for Inserting the package details 21](#_Toc21638773)

[9.1 Class Diagram 21](#_Toc21638774)

[9.2 PackageDAO.java 22](#_Toc21638775)

[9.3 PackageDAOImple.java 22](#_Toc21638776)

[10.0 Design for Updating the package details 23](#_Toc21638773)

[10.1 Class Diagram 23](#_Toc21638774)

[10.2 PackageDAO.java 24](#_Toc21638775)

[10.3 PackageDAOImple.java 24](#_Toc21638776)

[11.0 Design for Inserting the warehoue details 25](#_Toc21638773)

[11.1 Class Diagram 25](#_Toc21638774)

[11.2 WarehouseDAO.java 26](#_Toc21638775)

[11.3 WarehouseDAOImple.java 26](#_Toc21638776)

[12.0 Design for Validating the warehoue details 27](#_Toc21638773)

[12.1 Class Diagram 27](#_Toc21638774)

[12.2 WarehouseDAO.java 28](#_Toc21638775)

[12.3 WarehouseDAOImple.java 28](#_Toc21638776)

[13.0 Standards and Guidelines 29](#_Toc21638778)

[13.1 Java 29](#_Toc21638779)

[14.0 Submission 30](#_Toc21638780)

[14.1 Code submission instructions 30](#_Toc21638781)

[15.0 Change Log 31](#_Toc21638782)

# Introduction

## Purpose of this document

The purpose of this document is to define the Java class related implementation for Courier Tracking System project.

## Definitions & Acronyms

|  |  |
| --- | --- |
| Definition / Acronym | Description |
|  |  |

## Project Overview

Refer Courier Tracking System-use-case-specification.docx for understanding the functionality and features.

## Scope

Creation of model and data access object classes for Courier Tracking System application

## Intended Audience

* Product Owner
* Scrum Master
* Application Architect
* Project Manager
* Test Manager
* Development Team
* Testing Team

## Hardware and Software Requirement

1. Hardware Requirement:
   1. Developer PC with 4GB Ram
2. Software Requirement
   1. Git
   2. JDK 1.8
   3. Eclipse IDE for Enterprise Java Developers 2019-03 R

## Eclipse Project Configuration

The project cloned from Git needs to be set up as Eclipse project to make it easier with Java development. Find below the steps to configure Eclipse for Courier Tracking System project.

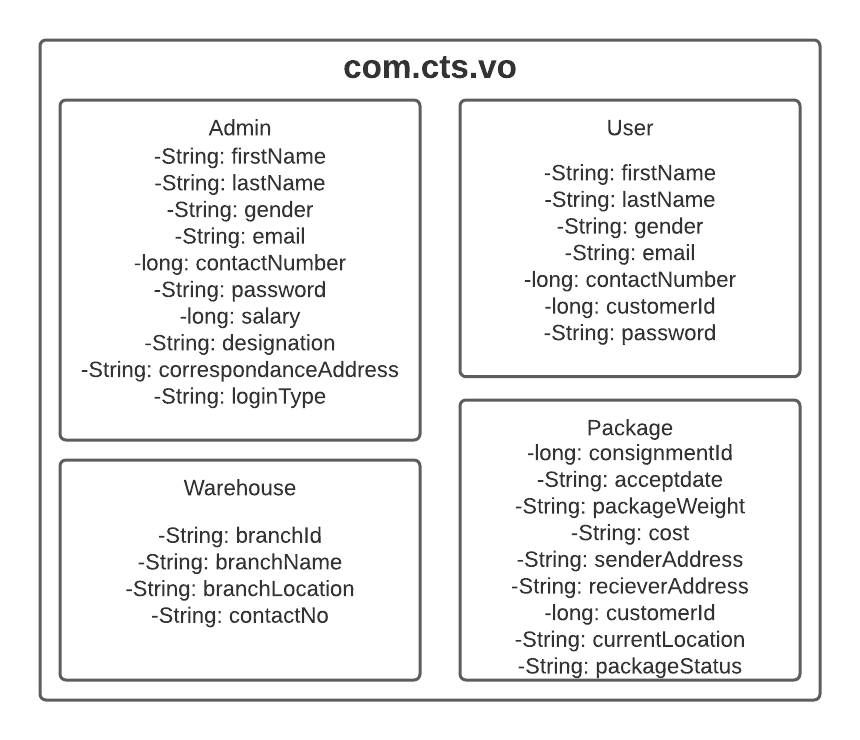
1. Open Eclipse
2. File > Open Projects from File System .. > Click “Directory..”
3. Select the Courier Tracking System folder in your PC where the code was cloned from <https://code.cognizant.com>
4. Click “Finish”, which will create the Courier Tracking System project and will be available in the project listed in the left hand side.
5. Now this project needs to be converted into Dynamic Web Project, so that it helps to work on Web Application. Find below the steps for converting the project into Dynamic Web Project:
   1. Right on the eKart project in the Project Explorer
   2. Properties > Project Facets > Convert to faceted form…
   3. Check “Dynamic Web Module” and “Java”.
   4. Click “Apply and Close”

# Class Diagram

The classes specified in this document are the primary Java classes that are required for implementation of Courier Tracking System application. Since JDBC module is covered later, the actual database implementation details are postponed to the respective module. The classes in this specification are implemented with hardcoded values and will be consumed by the Servlets when implementing the next module.

## VO package

Following are the real world objects identified for Courier Tracking System application. Admin refers to the details of the Admin and staff of the courier tracking system. User refers to the details of users registering on the portal. Warehouse and Package refers to the related details of them. Refer the diagram below and create classes accordingly.



Guidelines for understanding the above class diagram:

1. “com.cts.vo” represents the package
2. Admin, User, Package and Warehouse are classes
3. The content within all classes are instance variables
4. The hypen in each line represents private access specifier
5. For the sake of simplicity the constructors, getter and setter method are not included in the diagram. But it needs to be implemented in code. Code generation option in Eclipse can be used to generate code:
   1. Default constructor of the class.
   2. Constructor with option to set all instance variables.
   3. Getter and Setter method for each instance variable.

## Util Package

Common reusable classes and methods across Courier Tracking System application will be included in this package.



Guidelines for understanding the above class diagram:

1. “com.cts.util” represents the package.
2. Db is a class.
3. Underline denotes static method.

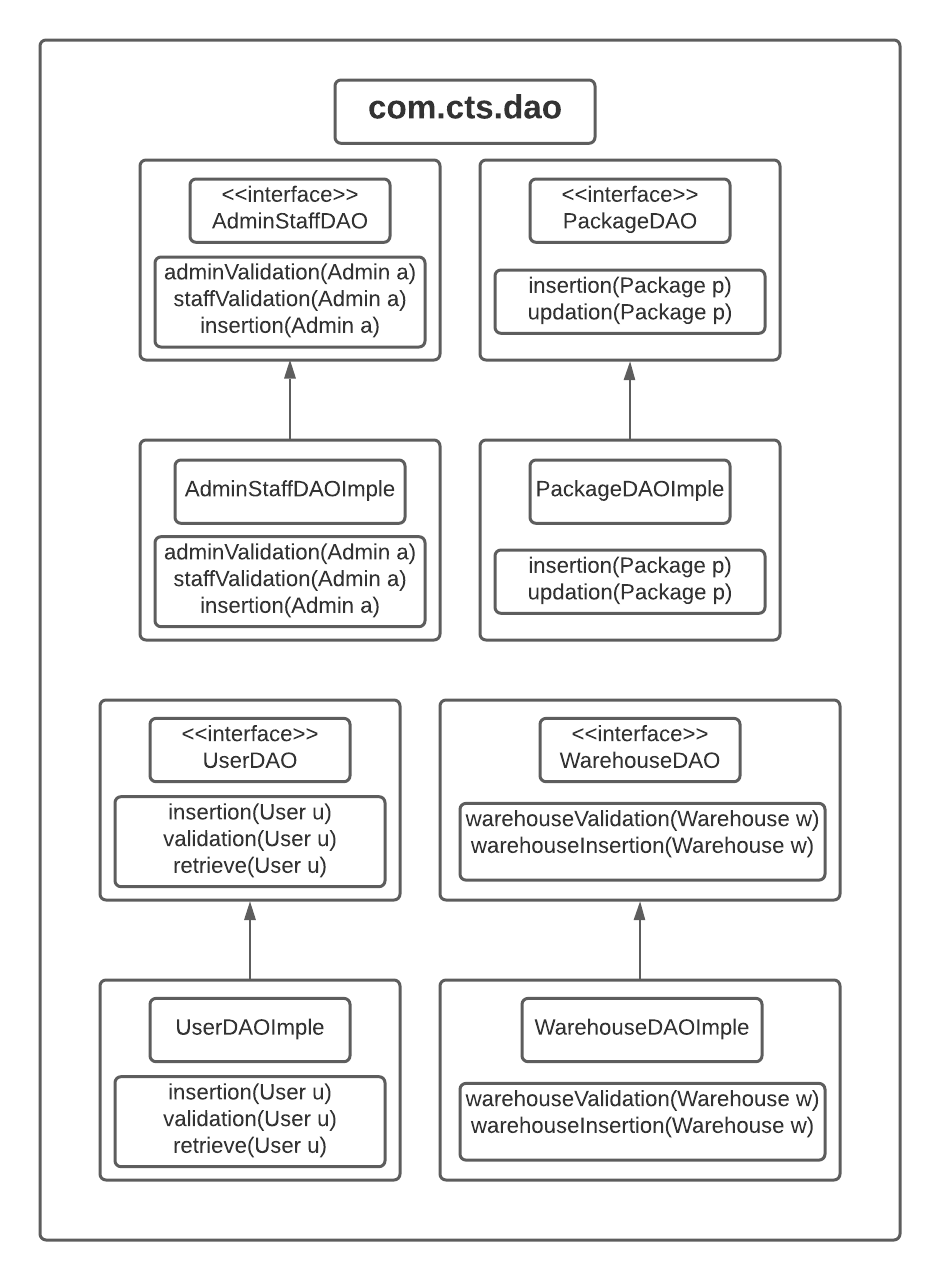
### Db.java

**Connection getDb():**

This method is used to get an instance of Connection Class and to establish a connection with the database.

1. Using Class.forName() method set mysql as JDBC Driver.
2. Using DriverManager class set connection with database in mysql.

## Dao package

This package contains the list of classes that will code to manage the data for Courier Tracking System application. The methods in Dao classes will be tested using their respective test classes. The Dao interface classes will act as a contract for working with any database. In this specification the implementation of AdminStaffDAOImple, PackageDAOImple, UserDAOImple, WarehouseDAOImple will be Collection framework based implementation of Dao interfaces AdminStaffDAO, PackageDAO, UserDAO and WarehouseDAO. 

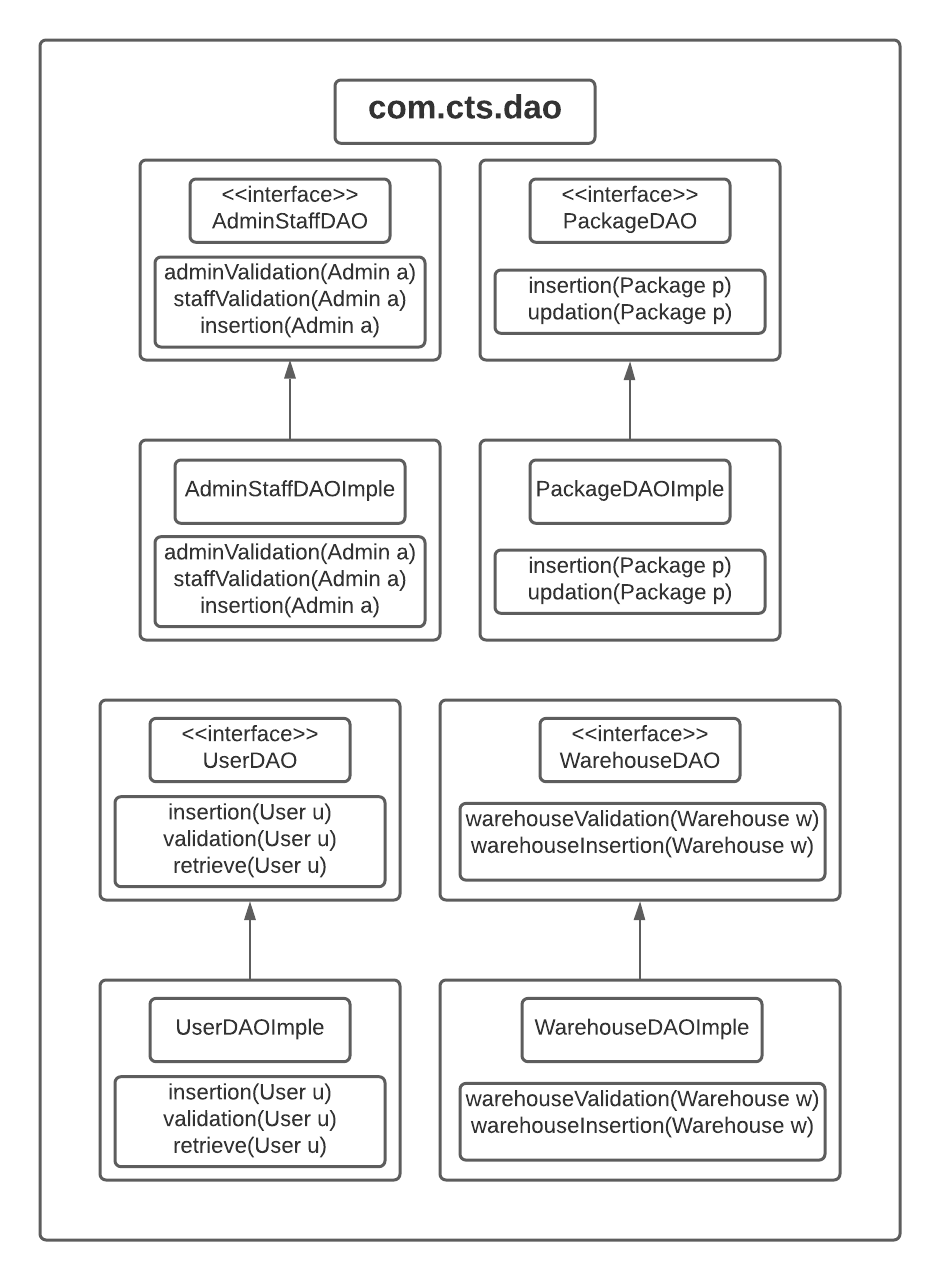
Guidelines for understanding the above class diagram:

1. Identify the package, classes, access modifiers, methods and static methods from the above diagram.
2. AdminStaffDAO, PackageDAO, UserDAO and WarehouseDAO are interfaces.
3. AdminStaffDAOImple, PackageDAOImple, UserDAOImple, WarehouseDAOImple are implementation classes for the interfaces as denoted by the arrow line.

# Design for Validation of Admin.

## Class Diagram

The below diagram denotes the methods that needs to be implemented for this use case. Method wise specification is defined after the diagram.



## AdminStaffDAO.java

Add the method public int adminValidation(Admin a)in the interface.

## AdminStaffDAOImple.java

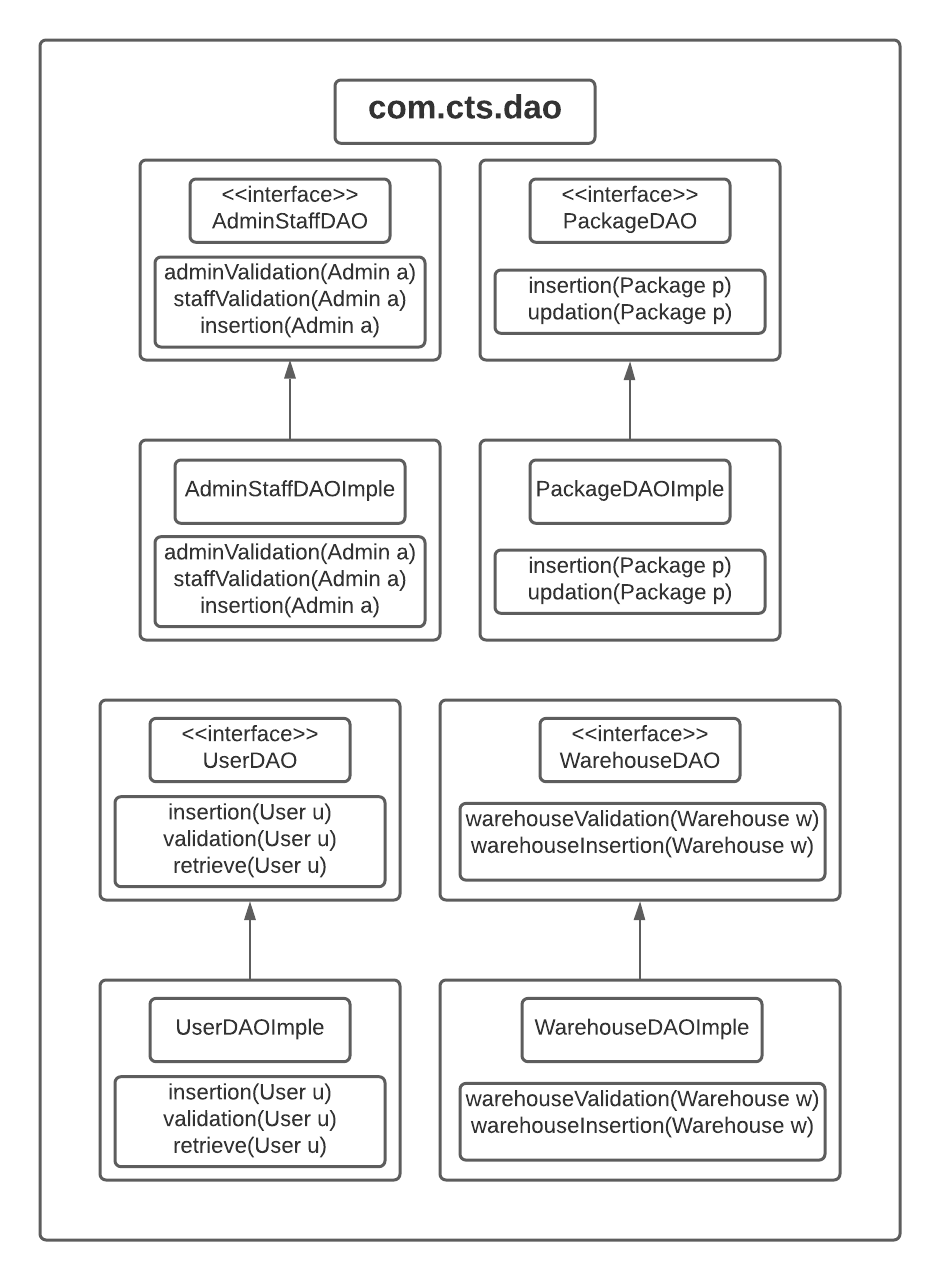
**public int adminValidation(Admin a)**

1. Retrieve all the data from staff table of cts database where loginType=’A’.
2. Compare the retrieved and user entered email and password.
3. Check the approvel status of the user and return an int value as follows:
   1. Return 1 if the request is “Approved”.
   2. Return 11 if the request is “Rejected”.
   3. Return 111 if the request is “Pending”.

# Design for Validation of Staff.

## Class Diagram

The below diagram denotes the methods that needs to be implemented for this use case. Method wise specification is defined after the diagram.



## AdminStaffDAO.java

Add the method public int staffValidation(Admin a)in the interface.

## AdminStaffDAOImple.java

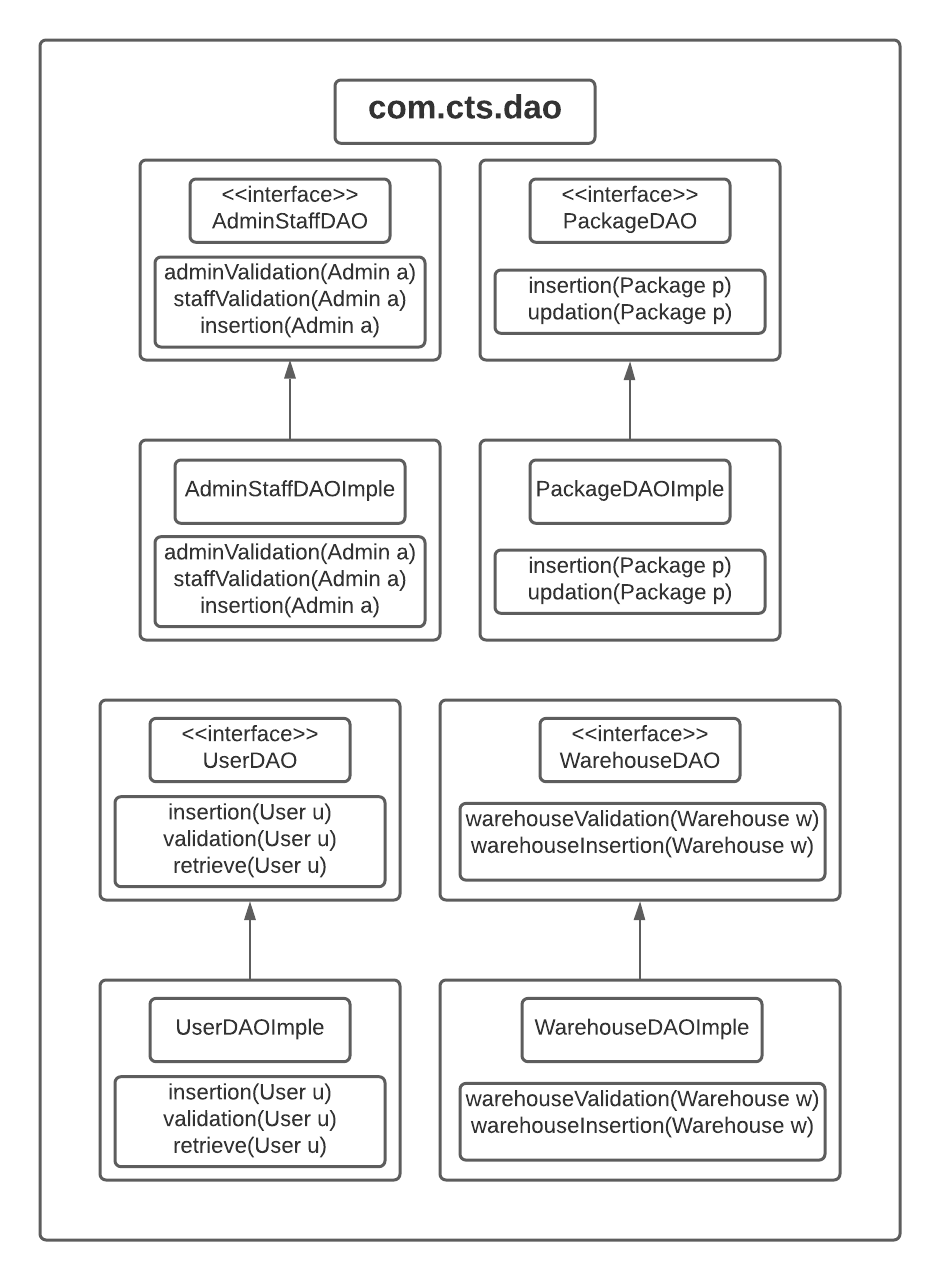
**public int staffValidation(Admin a)**

1. Retrieve all the data from staff table of cts database where loginType=’S’.
2. Compare the retrieved and user entered email and password.
3. Check the approvel status of the user and return an int value as follows:
   1. Return 2 if the request is “Approved”.
   2. Return 22 if the request is “Rejected”.
   3. Return 222 if the request is “Pending”.

# Design for Registration of Admin and Staff.

## Class Diagram

The below diagram denotes the methods that needs to be implemented for this use case. Method wise specification is defined after the diagram.



## AdminStaffDAO.java

Add the method public int insertion(Admin a)in the interface.

## AdminStaffDAOImple.java

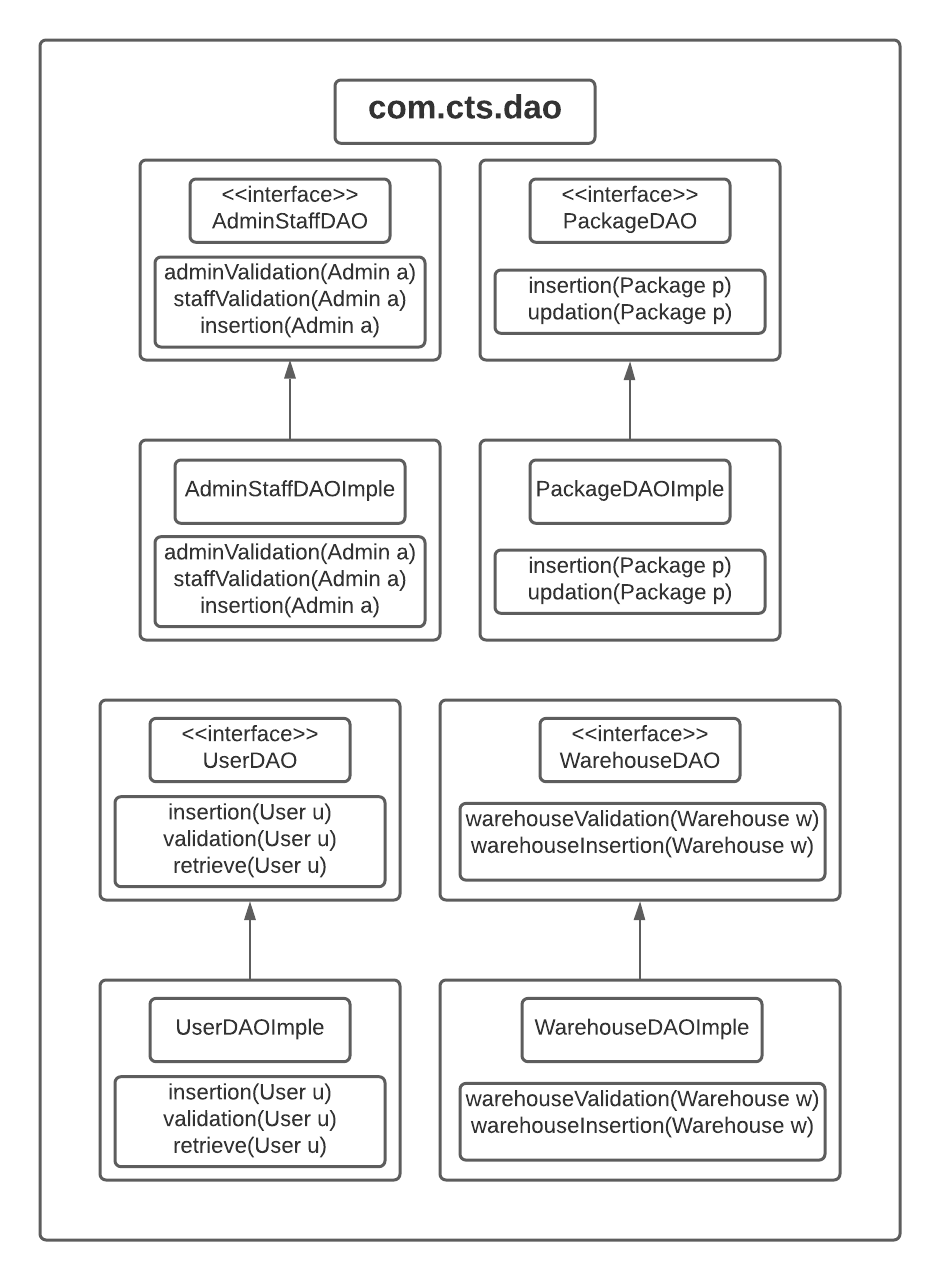
**public int insertion(Admin a)**

1. Insert all the data into staff table of cts database.
2. Execute the query and return the corresponding int value.

# Design for Registration of User.

## Class Diagram

The below diagram denotes the methods that needs to be implemented for this use case. Method wise specification is defined after the diagram.



## UserDAO.java

Add the method public int insertion(User u)in the interface.

## UserDAOImple.java

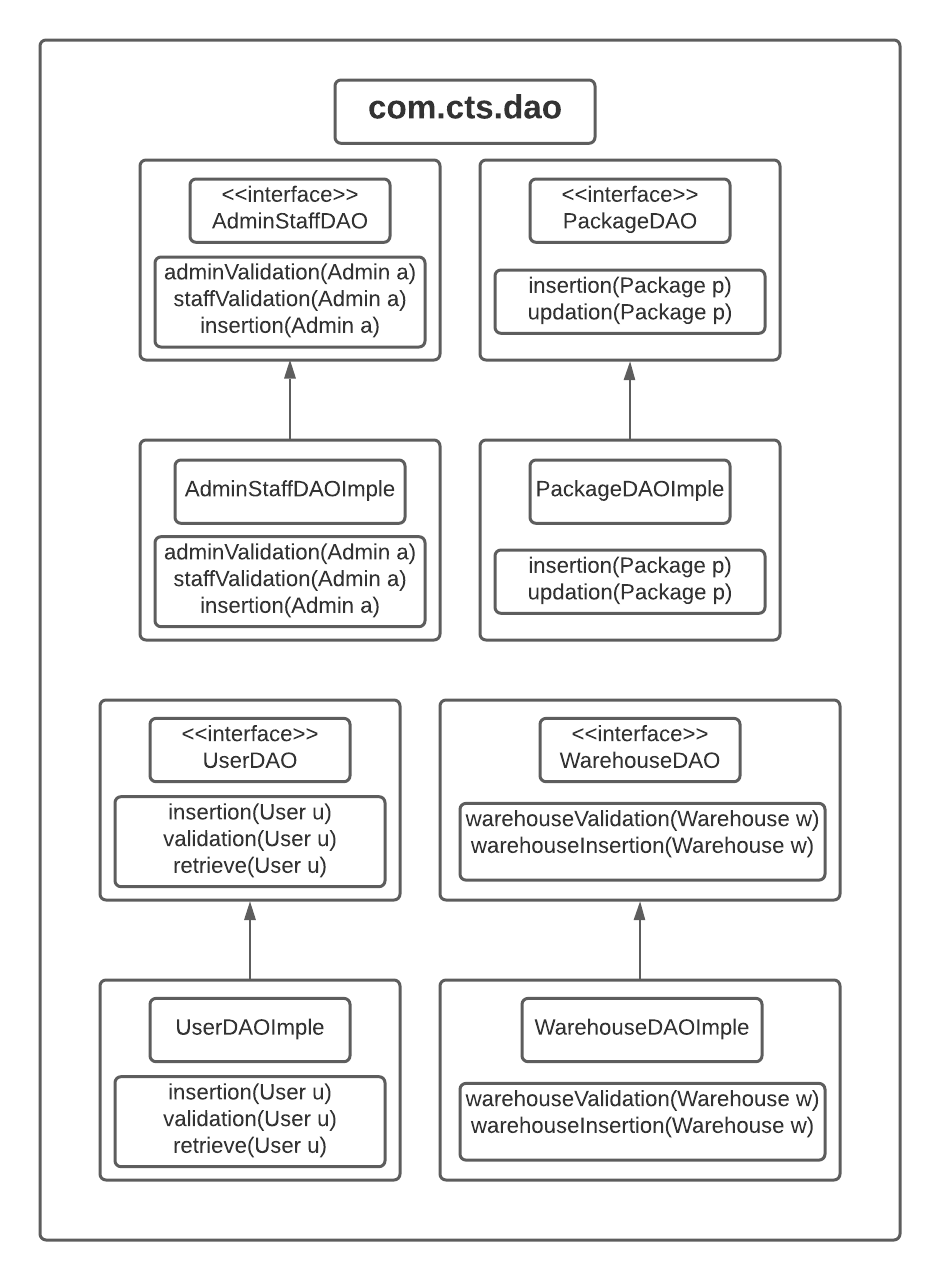
**public int insertion(Admin a)**

1. Insert all the data into user table of cts database.
2. Execute the query and return the corresponding int value.

# Design for Validation of User.

## Class Diagram

The below diagram denotes the methods that needs to be implemented for this use case. Method wise specification is defined after the diagram.



## UserDAO.java

Add the method public int validation(User u)in the interface.

## UserDAOImple.java

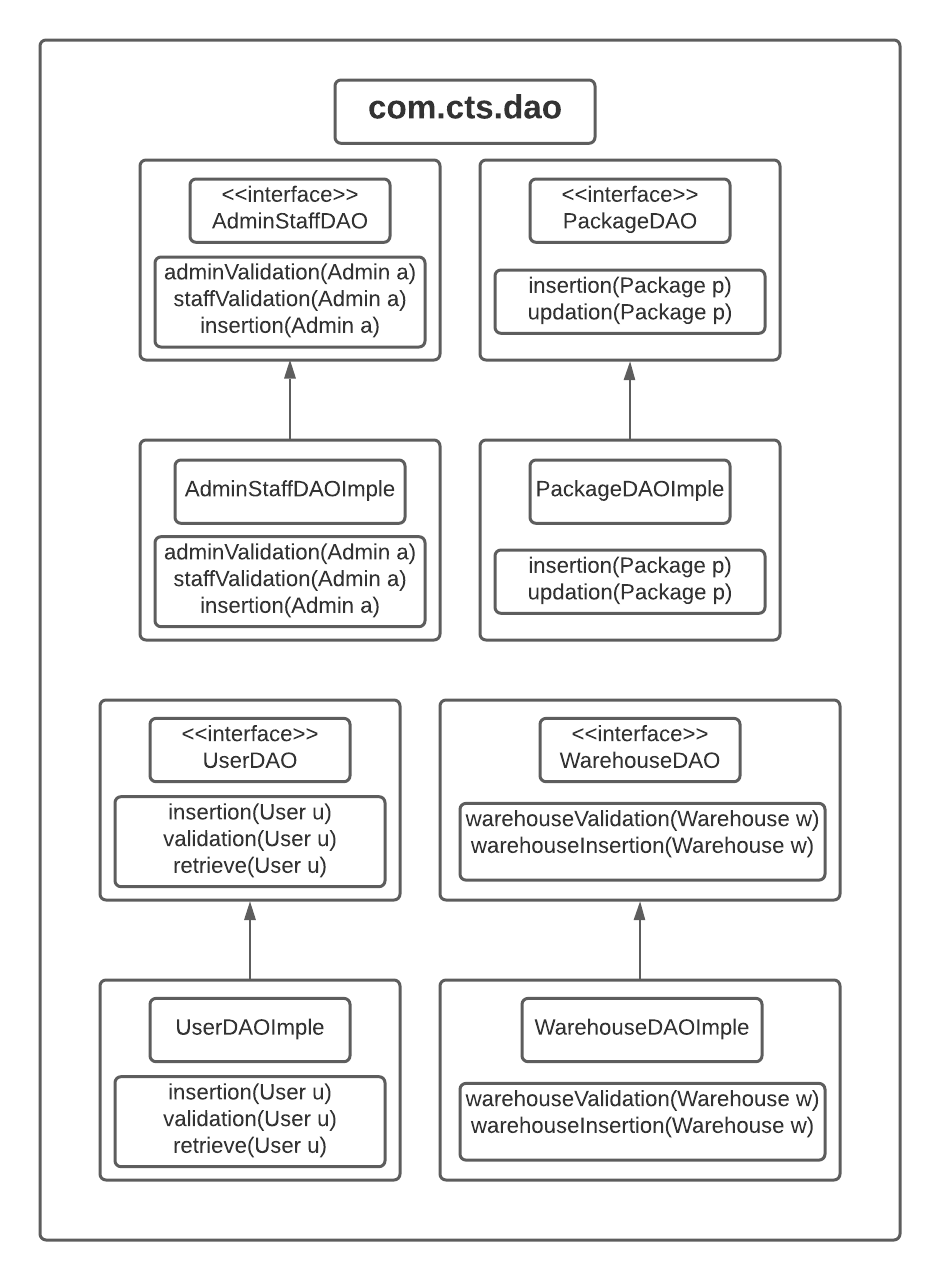
**public int validation(User u)**

1. Retrieve all the data from user table of cts database.
2. Compare the retrieved and user entered email and password.
3. Execute the query and return the corresponding int value.

# Design for Retrieving the package details of User.

## Class Diagram

The below diagram denotes the methods that needs to be implemented for this use case. Method wise specification is defined after the diagram.



## UserDAO.java

Add the method public ArrayList<Package> retrieve(User u)in the interface.

## UserDAOImple.java

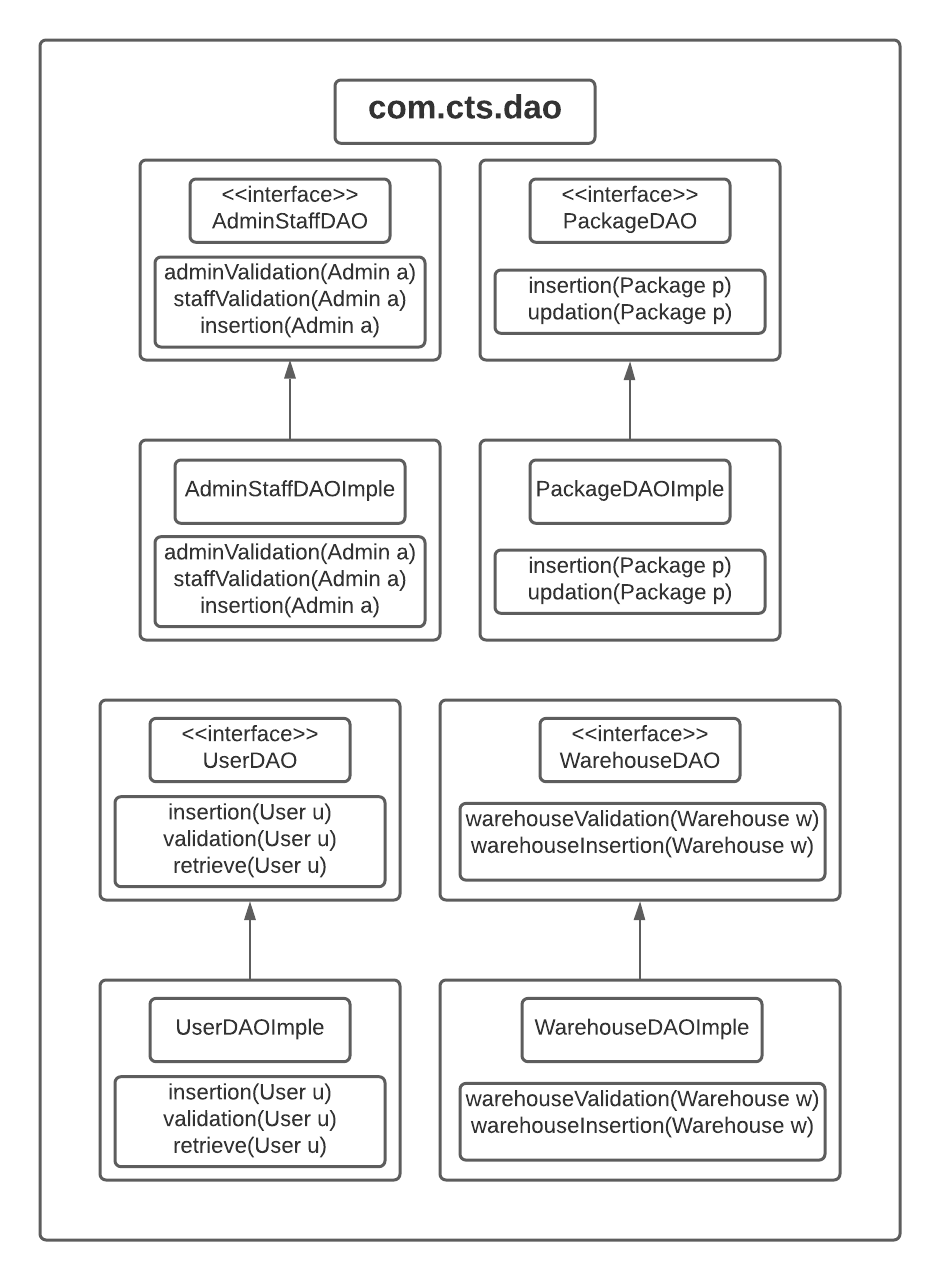
**public ArrayList<Package> retrieve(User u)**

1. Retrieve the data from package table of cts database where customerId is equals to the user entered customerId.
2. Add all the data to a package object through constructor injection.
3. Add the package object to an ArrayList and return that ArrayList.

# Design for Inserting the Package Details.

## Class Diagram

The below diagram denotes the methods that needs to be implemented for this use case. Method wise specification is defined after the diagram.



## PackageDAO.java

Add the method public int insertion(Package p)in the interface.

## PackageDAOImple.java

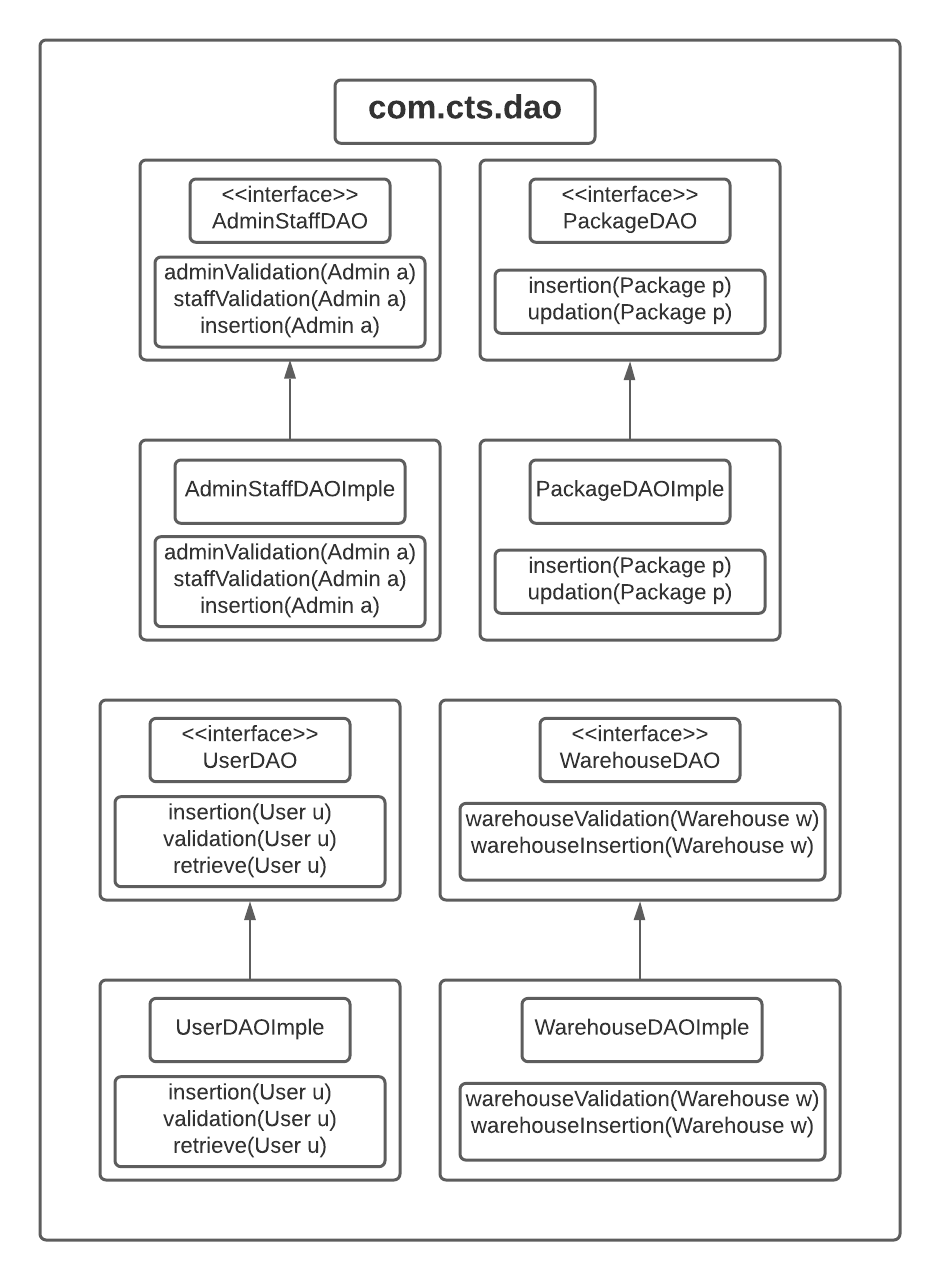
**public int insertion(Package p)**

1. Insert all the data into package table of cts database.
2. Execute the query and return the corresponding int value.

# Design for Updating the Package Details.

## Class Diagram

The below diagram denotes the methods that needs to be implemented for this use case. Method wise specification is defined after the diagram.



## PackageDAO.java

Add the method public int updation(Package p)in the interface.

## PackageDAOImple.java

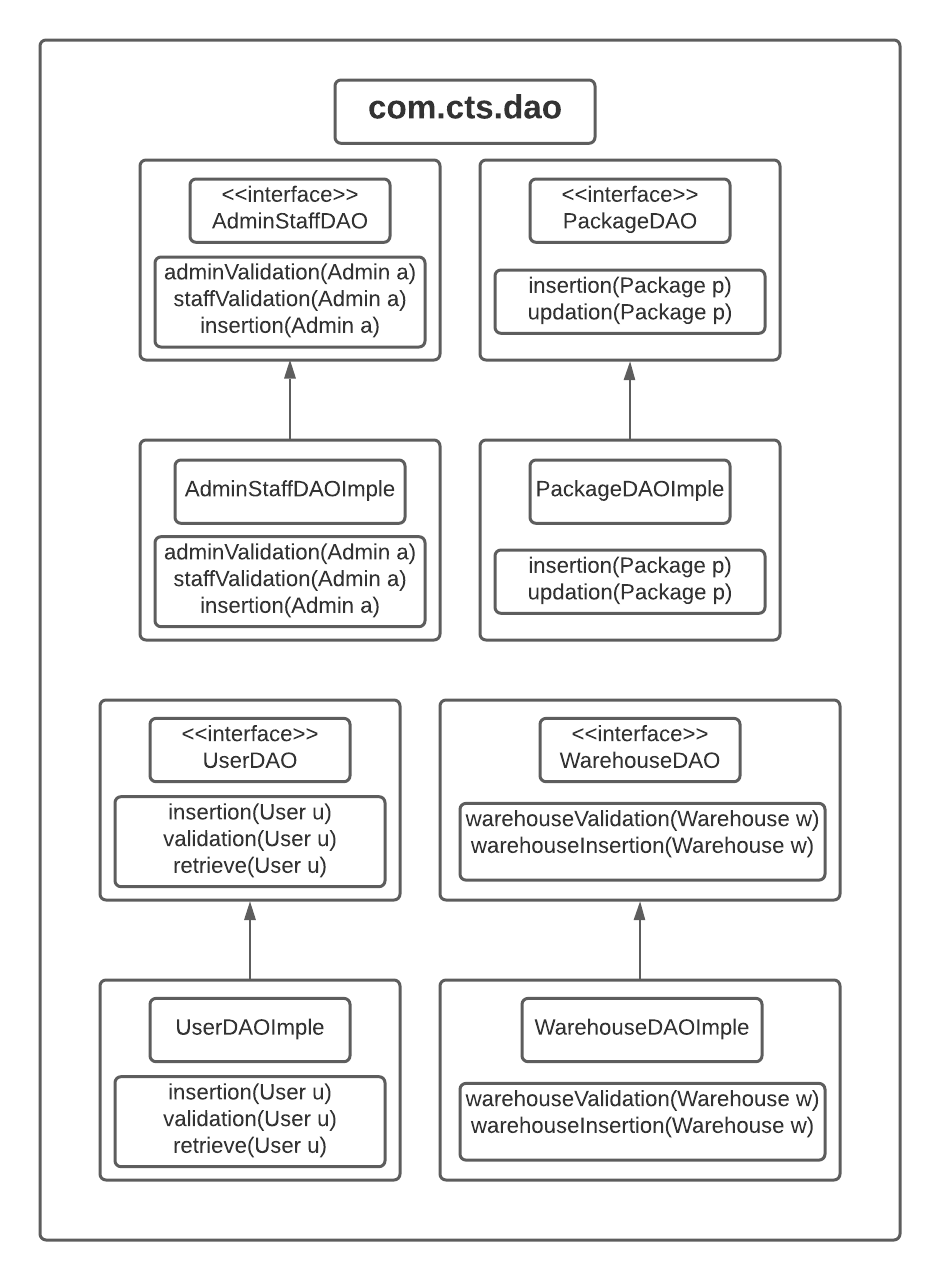
**public int updation(Package p)**

1. Update the corresponding changed details of the package into package table of cts database.
2. Execute the query and return the corresponding int value.

# Design for Inserting the Warehouse Details.

## Class Diagram

The below diagram denotes the methods that needs to be implemented for this use case. Method wise specification is defined after the diagram.



## WarehouseDAO.java

Add the method public int warehouseInsertion(Warehouse w)in the interface.

## WarehouseDAOImple.java

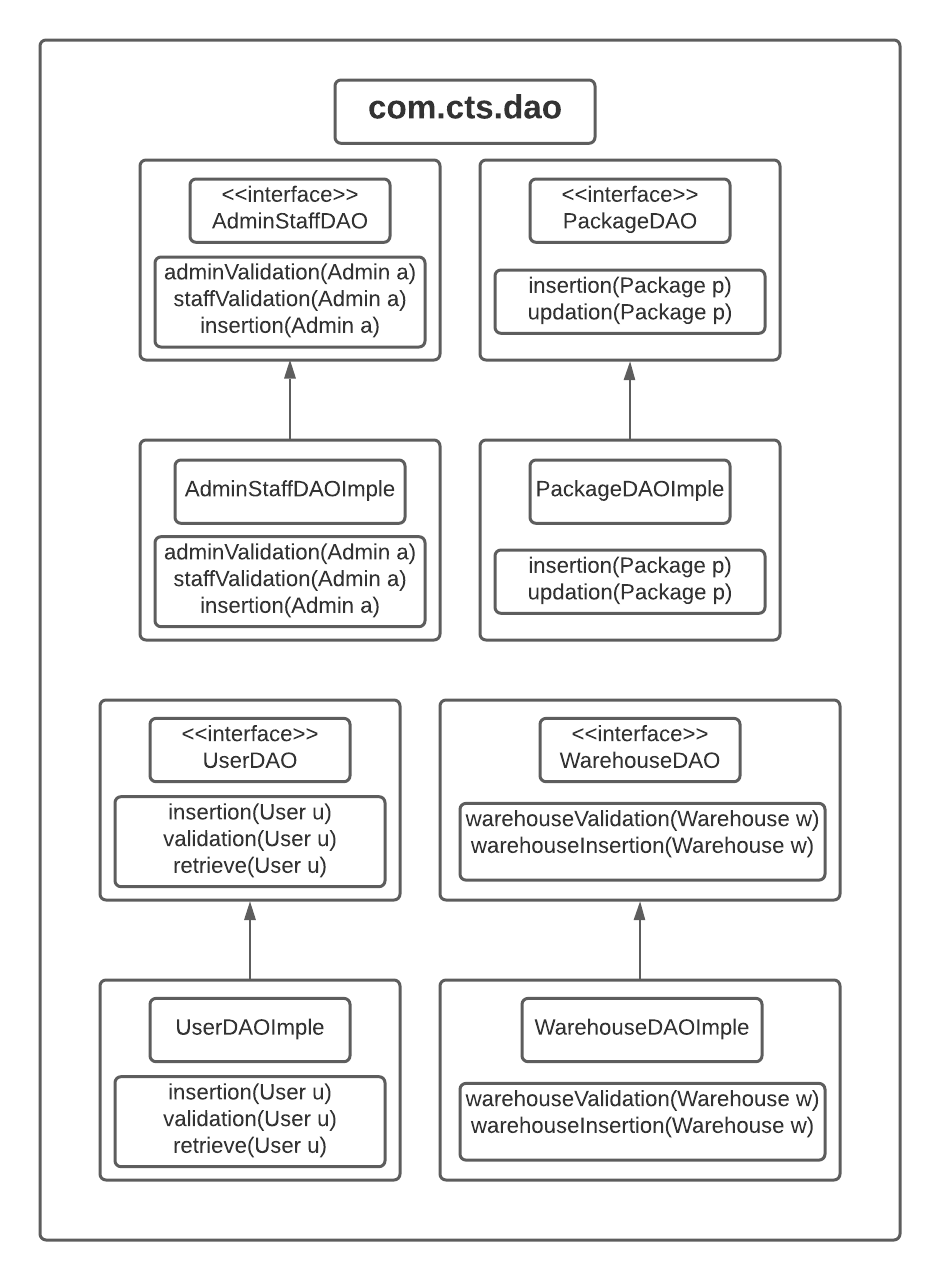
**public int warehouseInsertion(Warehouse w)**

1. Insert all the data into warehouse table of cts database.
2. Execute the query and return the corresponding int value.

# Design for Validating the Warehouse Details.

## Class Diagram

The below diagram denotes the methods that needs to be implemented for this use case. Method wise specification is defined after the diagram.



## WarehouseDAO.java

Add the method public int warehouseValidation(Warehouse w)in the interface.

## WarehouseDAOImple.java

**public int warehouseValidation(Warehouse w)**

1. Retrieve all the data from warehouse table of cts database.
2. Compare the retrieved and user entered branchId.
3. Execute the query and return the corresponding int value.

# Standards and Guidelines

## Java

1. Ensure that the class names, method names and variable names are followed exactly as specified in the class diagram
2. Ensure that access modifiers are in line with the class diagram specification
3. Naming standards to be followed:
   1. Variable
      1. Should be in mixed case with the first letter lowercase and with the first letter of each internal word capitalized (Example: firstName, lastName)
      2. Variable names should be short, but meaningful
      3. Variable name defined should indicate the purpose to a casual observer
      4. Single character variable names should be avoided except for temporary variables
      5. Temporary variables include u, p, a and w.
   2. Class
      1. Class name should be a noun
      2. Class name should be in mixed case with the first letter uppercase and with the first letter of each internal word capitalized
      3. Must use whole words and should not have acronyms or abbreviations

Examples: Package, Warehouse.

* 1. Method
     1. Method names should be verbs
     2. Method names should be in mixed case with the first letter lowercase and with the first letter of each internal word capitalized

Example: warehouseValidation().

1. Code Formatting
   1. Class Structure
      1. Place the elements of a class in the following order:
         1. Static variables
         2. Instance variables
         3. Constructors
         4. Methods and Getter/Setters
         5. hashCode(), equals(), toString().

* 1. Spacing
     1. A space before and after an operator is required
     2. A space before curly braces is required
     3. A space after a comma is required
     4. A space after semicolon in for loop is required
     5. A single line space after a method is required
  2. Curly braces position
     1. Opening curly braces should be in the same line
     2. Closing curly braces should always be in a new line
  3. Tab spacing
     1. Use 4 spaces instead of tab character
     2. Increase a tab character in the lines after opening curly braces
     3. Reduce a tab character on the of closing curly braces
     4. Include one more tab in the wrapped line
  4. Line Width
     1. Width of a line should not exceed 100 characters

# Submission

## Code submission instructions

Once your code is evaluated by the trainer and all the issues reported by the trainer are corrected, the code needs to be submitted to the remote repository. Follow the steps below to submit the code to remote repository.

1. In Windows Explorer go to the Courier Tracking System folder
2. Right click on the empty space in the right hand side of Windows Explorer and select “Git Bash here”
3. Execute the following commands

To display the added or modified files

git status

To stage the added or modified files

git add .

To display the staged files

git status

To save the code to local repository

git commit -m "java"

To transfer the changes from local machine to server

git push origin master

1. Successful execution of the above commands will upload the files to the server repository.
2. Login into <https://code.cognizant.com>
3. Click on the project Courier Tracking System
4. Check if the files that are uploaded correctly with appropriate folder structure.

# Change Log

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Changes Made | | | |
| V1.0.0 | Initial baseline created on <dd-Mon-yy> by <Name of Author> | | | |
| Vx.y.z | <Please refer the configuration control tool / change item status form if the details of changes are maintained separately. If not, the template given below needs to be followed> | | | |
| **Section No.** | **Changed By** | **Effective Date** | **Changes Effected** |
|  |  |  |  |