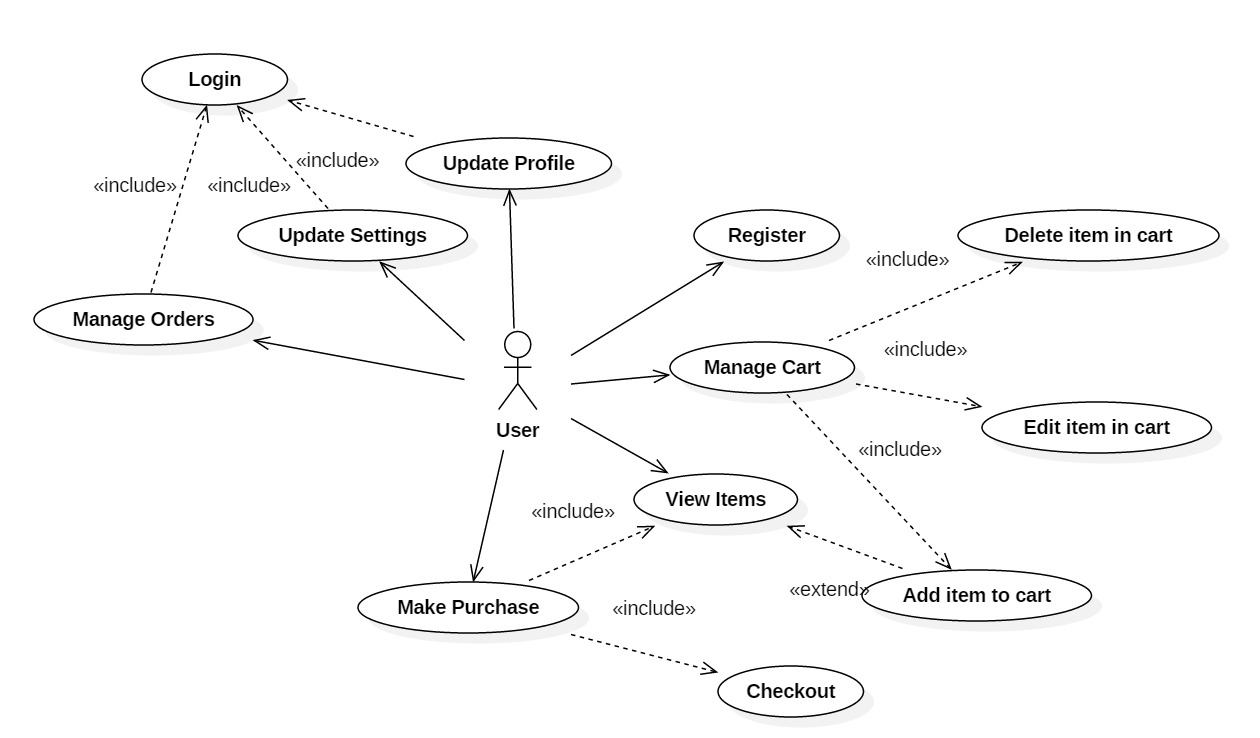
Shopping Management System

# User and Purchase Management Subsytem

## Use case diagram

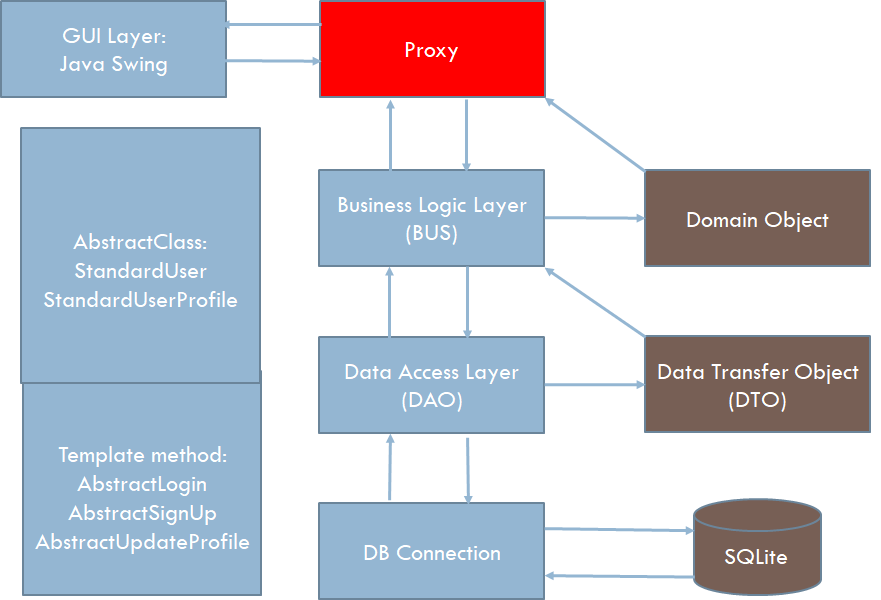


Two parts are included:

Part 1 is user management which contains register, login, update profile and update settings.

Part 2 is purchase management containing manage orders, make purchase, view items and manage cart.

## Subsystem Architecture



GUI Layer: To provide graphical user interface for system users

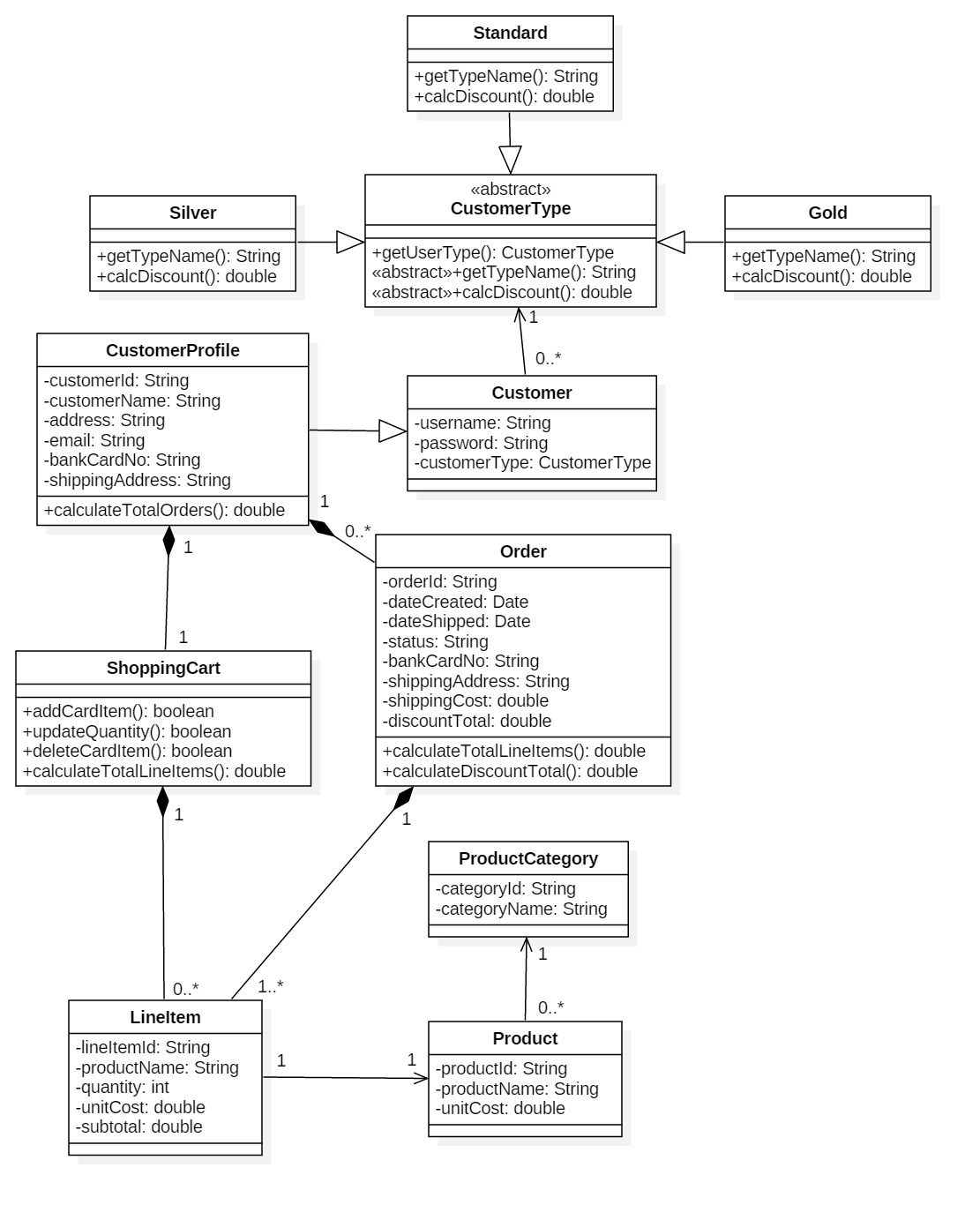
Proxy: To control access of user to system

BUS: To model business behavior interaction with domain object and data transfer object (DTO)

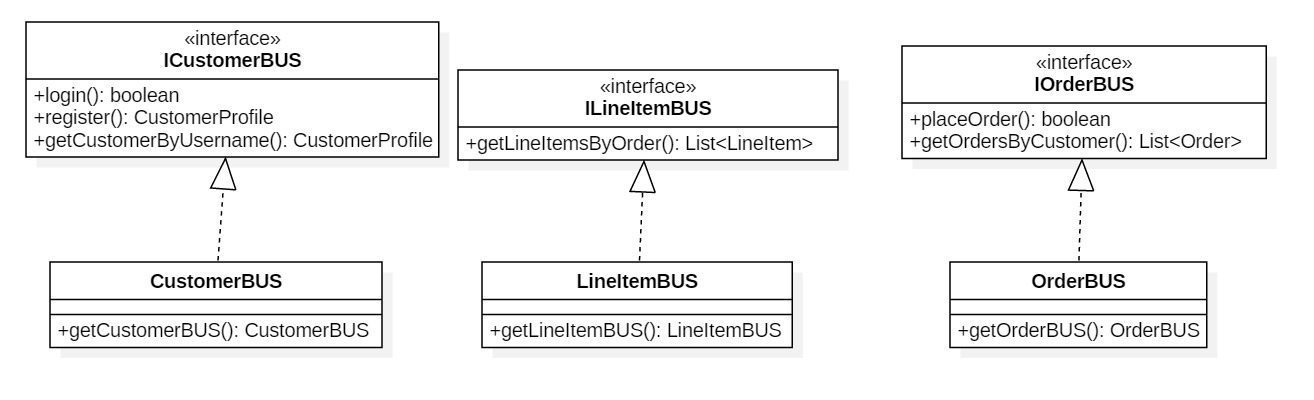
DAO: To manage data access between data transfer object (DTO) and data base

DB Connection: To provide connection between DAO and data base

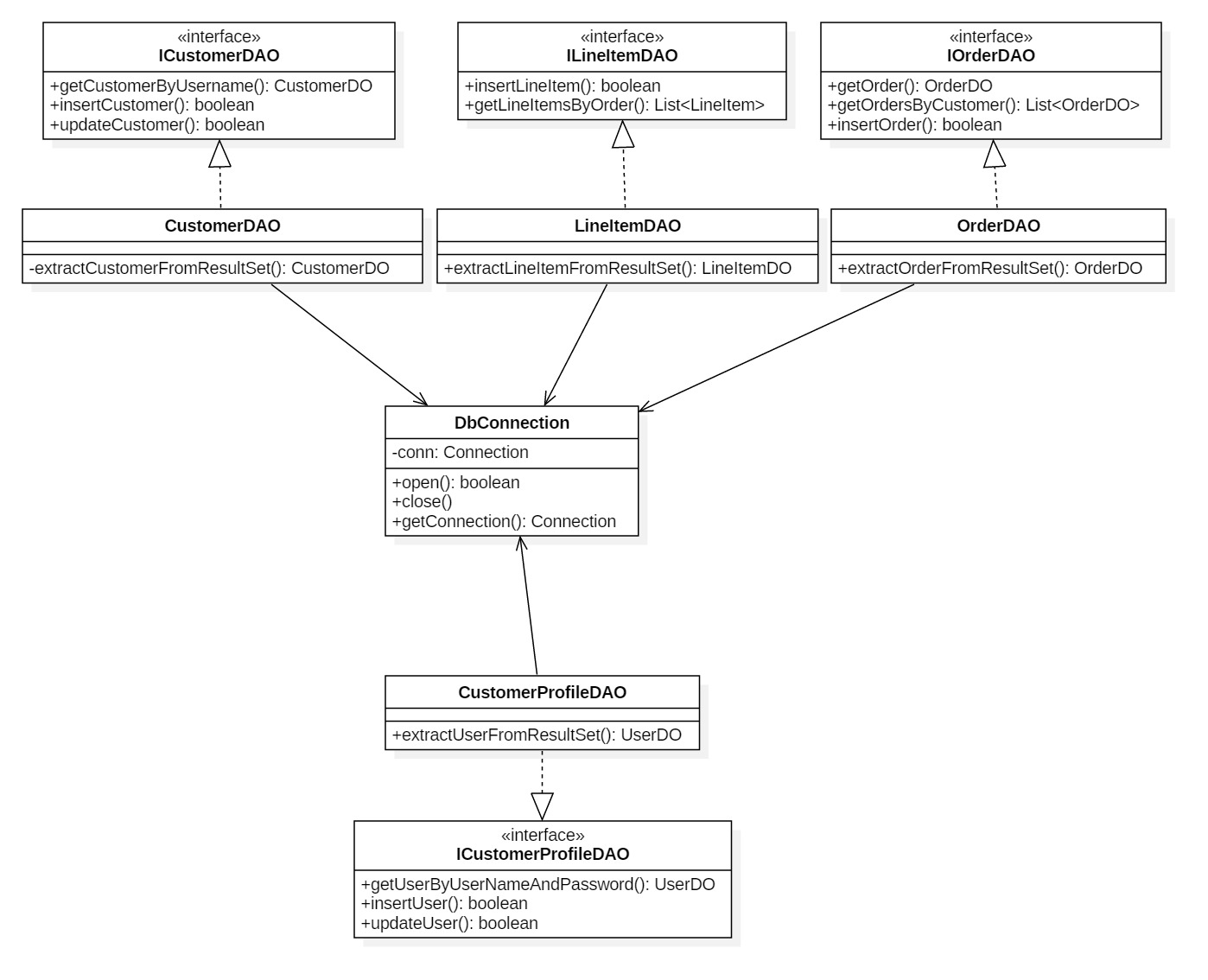
## Class diagram for Domain



## Class diagram for Business Layer

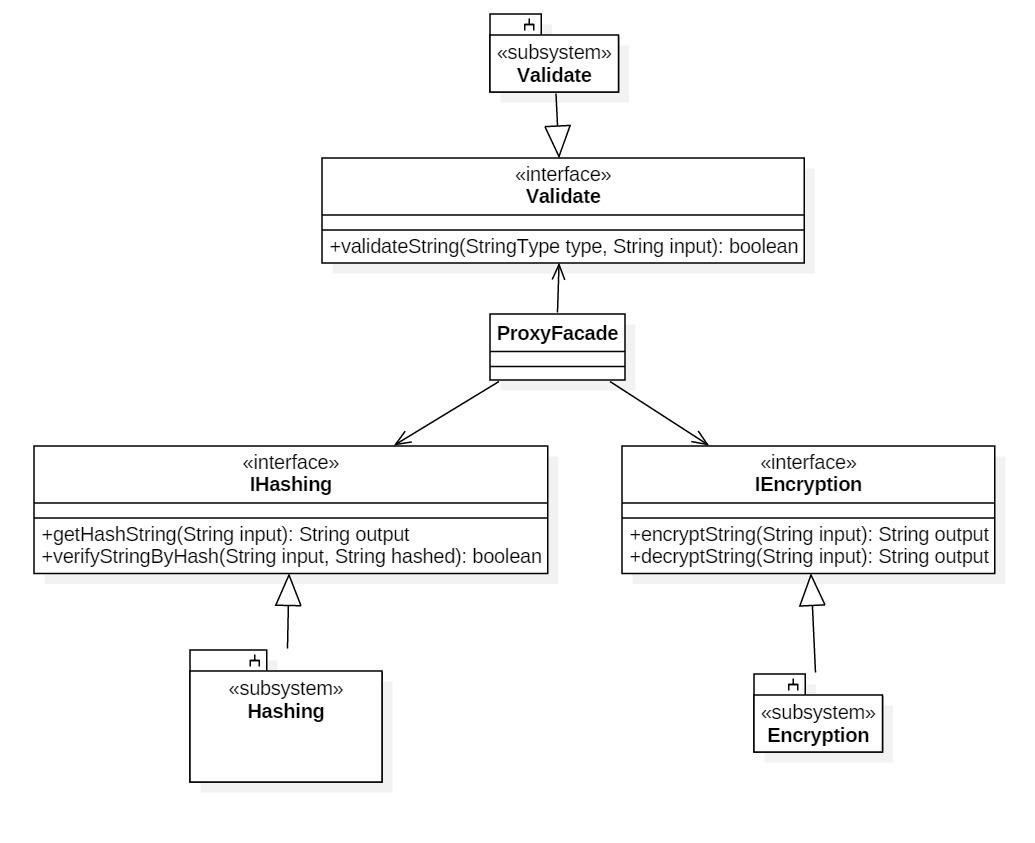


## Class diagram for DAO Layer



# Membership Framework

## Class diagram for Proxy Layer



# Product Management Subsystem

## Use case diagram

Five use cases are included:

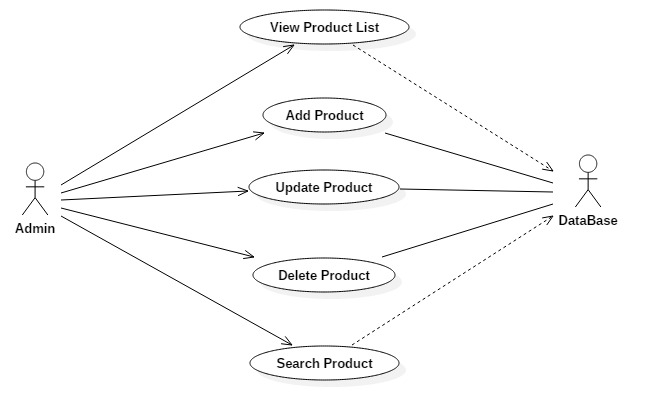
Admin user views product List

Admin user adds product

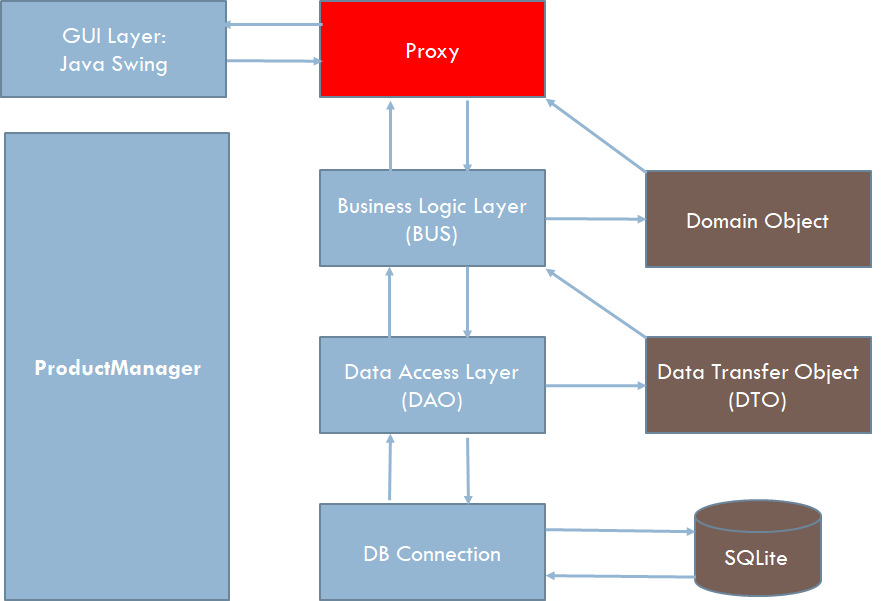
Admin user updates product

Admin user deletes product

Admin user searches product by product name or category

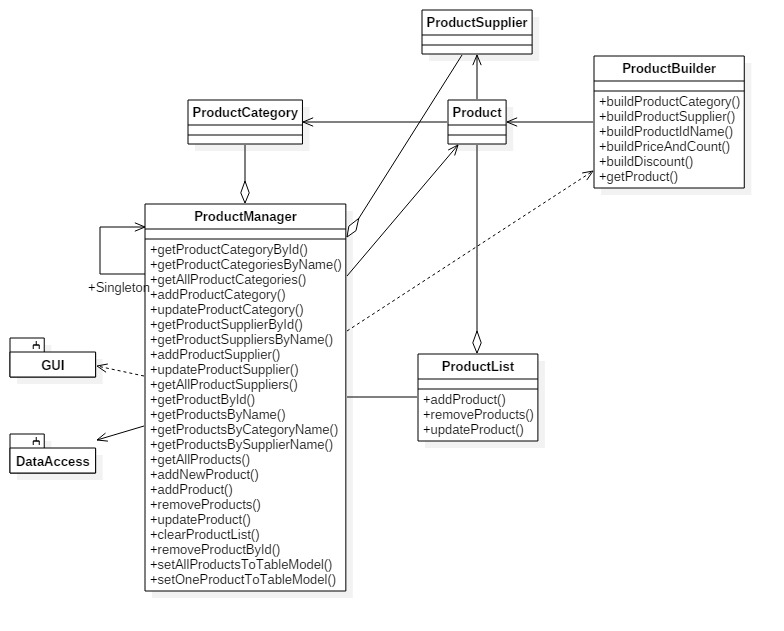


## Subsystem Architecture



Same work flow structure as user and purchase management subsystem. Whereas, using a ProductManager to manage the whole control flow.

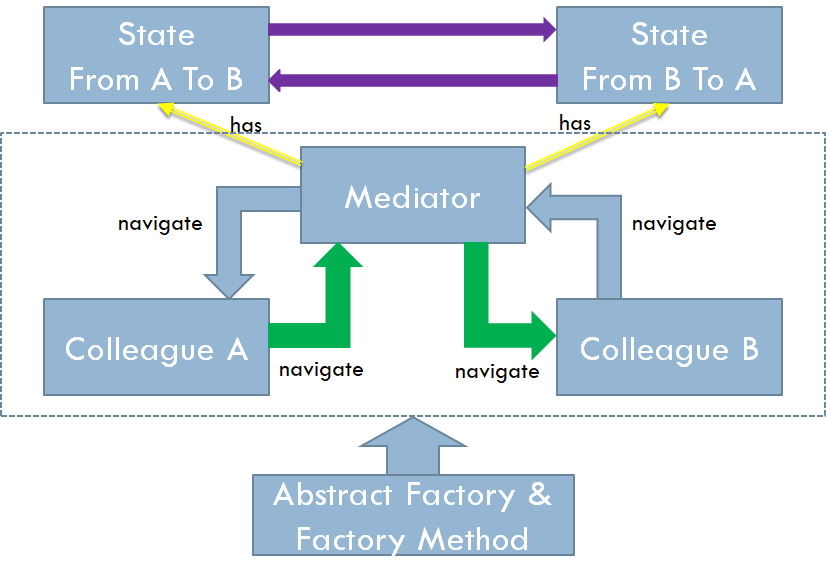
## Class diagram for ProductManager



# GUI Navigation Framework

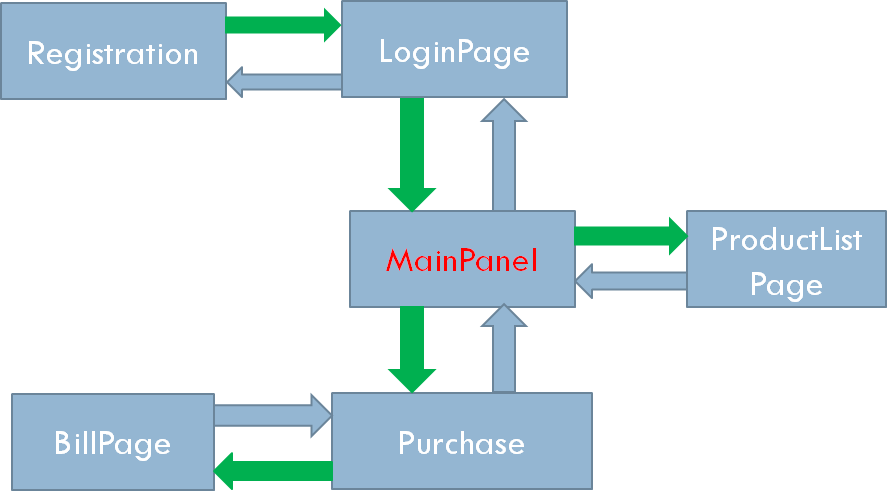
## Framework structure diagram

Abstract Factory Pattern, Factory Method Pattern, Mediator Pattern as well as State Pattern build the core skeleton of the framework.



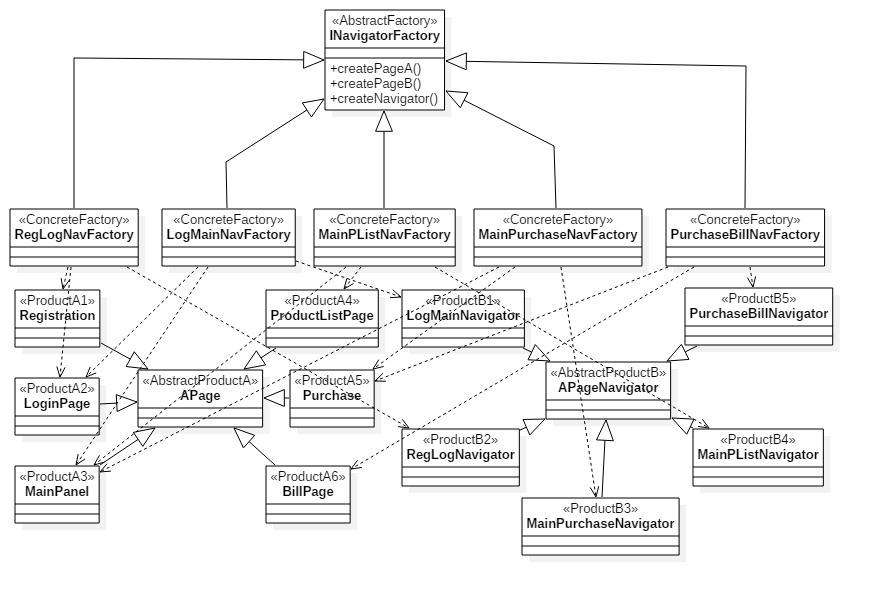
## Framework Application Shopping Management System

Six concrete colleaques are separated into five pairs of navigation which are managed by five independent concrete mediators.

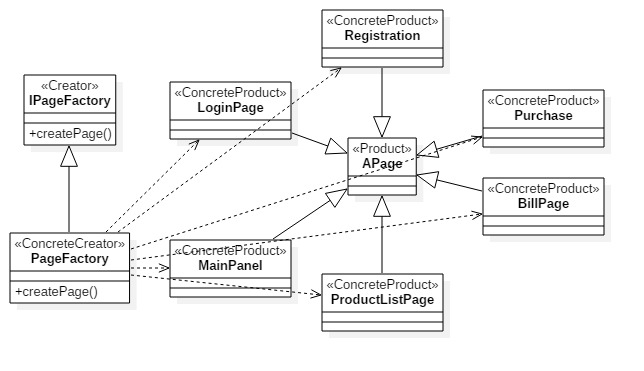


## Framework Class Diagrams in Application

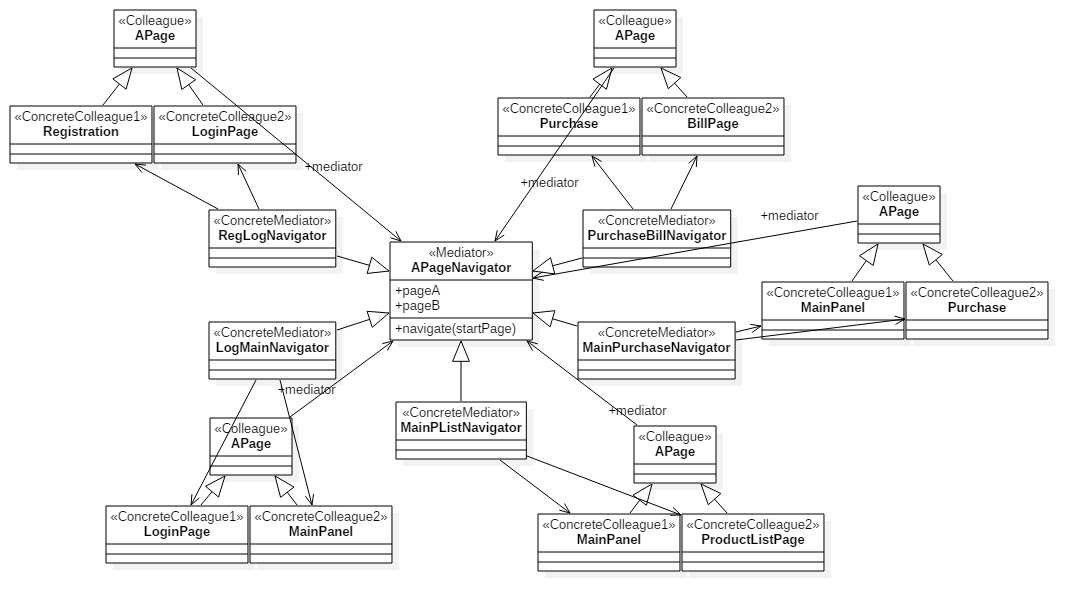
* Abstract Factory



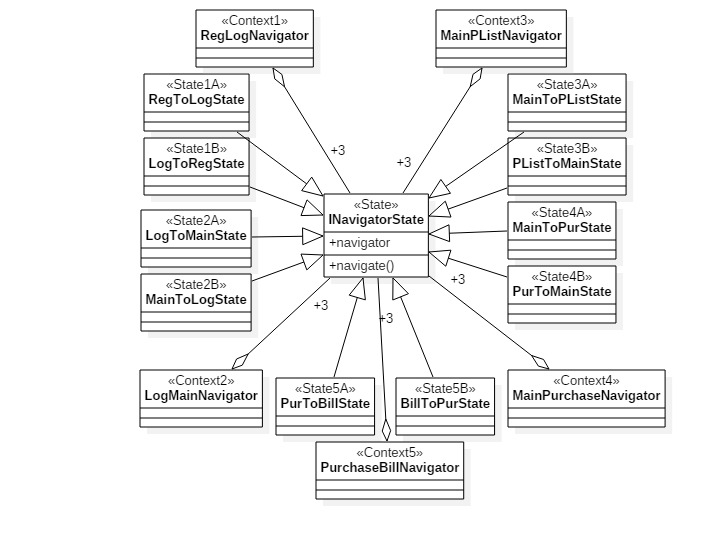
* Factory Method



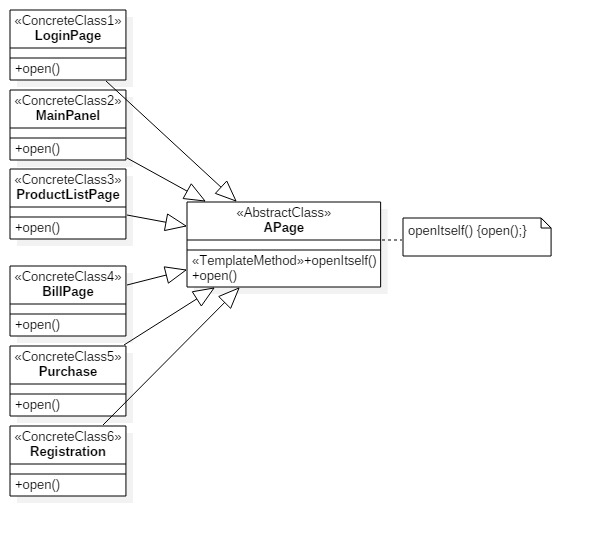
* Mediator



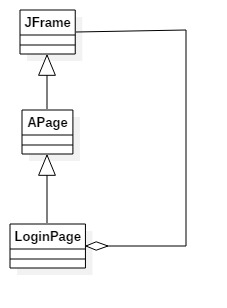
* State



* Template Method



* Composite



# Recommendation and Data Access Framework

## Framework user cases diagram

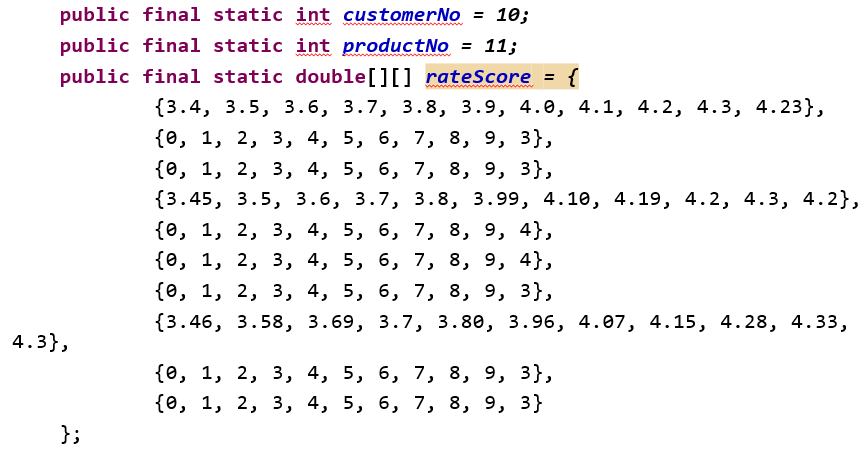


Recommendation system could be one layer of service independ of any external components, for example, constrcut it based on Spring boot and provide rest-api service.

It provides registered interface, historic data loading interface and output interface.

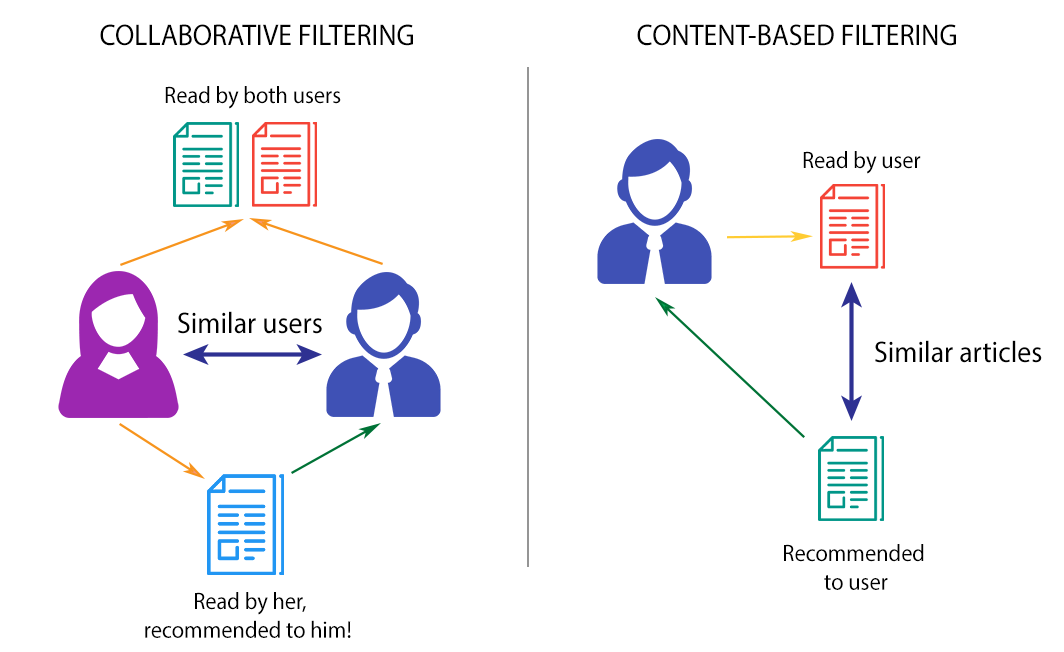
There are two algorithms to calculate the recommendation,one is Collaborative filtering and the other is Co-Occurrence.

**Collarborative Filtering:**

****

As the above maxtrix shows: For maxtrix rateScore, the row stands for the cusomer, the col stands for the product, so the rateScore[1][2] means the review score of custoemr 1 for product 2. The customer can be called similar if they have similar review scores for the same product. We can recommend to one customer the products that his/her similar customers like.

Collaborative filtering, also referred to as social filtering, filters information by using the recommendations of other people. It is based on the idea that people who agreed in their evaluation of certain items in the past are likely to agree again in the future. A person who wants to see a movie for example, might ask for recommendations from friends. The recommendations of some friends who have similar interests are trusted more than recommendations from others. This information is used in the decision on which movie to see.

[](https://www.google.com/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwid3rOI3JfZAhXM5oMKHUrvCHsQjRwIBw&url=https://www.themarketingtechnologist.co/building-a-recommendation-engine-for-geek-setting-up-the-prerequisites-13/&psig=AOvVaw2sU8xwR30JLbKdBb737pR-&ust=1518227430709587)

The neighborhood-based algorithm calculates the similarity between two users or items, and produces a prediction for the user by taking the weighted average of all the ratings. Similarity computation between items or users is an important part of this approach. Multiple measures, such as Pearson correlation and vector cosine based similarity are used for this.

**Co-occurrence:**

Co-occurrence means two or more products occurred in one customer purchase records.

Customer1: book1 book2 book3 book5 book6 book8 book2

Customer2: book3 book4 book6 book8 book2 book3 book9

Customer3: book2 book3 book4 book5 book6 book8 book7

So it’s a high possibility that one customer could buy B after buy A if most customers done so far.

## Framework user cases diagram



Both product management and purchase will use data access Framework to access database.

## The design pattern used in recommendation and data access framework

## 5.1 Singleton

**public** **static** RateInformation getRateInformation(){

**if**(*instance* == **null**){

**synchronized** (RateInformation.**class**) {

**if**(*instance* == **null**){

*instance* = **new** RateInformation();

}

}

}

**return** *instance*;

}

/\*\*

\* The Singleton Constructor. Note that it is private! No client can

\* instantiate a Singleton object directly!

\*/

**private** RateInformation() {}

**public** **void** assignRateInformation(**int** customerCount, **int** procductCount, **double** scores[][]) {

**this**.customerCount = customerCount;

**this**.procductCount = procductCount;

rateScores = **new** **double**[customerCount][procductCount];

**for**(**int** row = 0; row<customerCount; row++)

**for**(**int** col = 0; col<procductCount; col++)

rateScores[row][col] = scores[row][col];

assigned = **true**;

}

## 5.2 Adapter



Since the class JDBCAccessFacade is developed after the ProductDao, so to use the JDBCAccessFacade, we could transform the interface of ProductDao to JDBCFacade by using the Adapter patterns.

## 5.3 Façade

****

The class JDBCAccessFacade is developed to access the database related operations except Select operations. It encapsulates Create, Delete and Update operations include the parameterized prepared SQL.

## 5.4 Strategy



Suppose we get m most bigger ones from one unsorted array.

If m is small comparing to n, we can easily scan the array m times, so the time complexity is (m\*n) which approximately to n.

Otherwise if m is close to n, if we still scan m times, the time complexity is changed to n\*n.

How about do quicksort on the array and then get the most values ones? The time complexity is O(nlogn). I finally considered to use the max heap. The heap build time is O(n), each time we got one object from heap and then rebuild it. The total time complexity is O(n+mLogn). This might be the best method I could thought out.

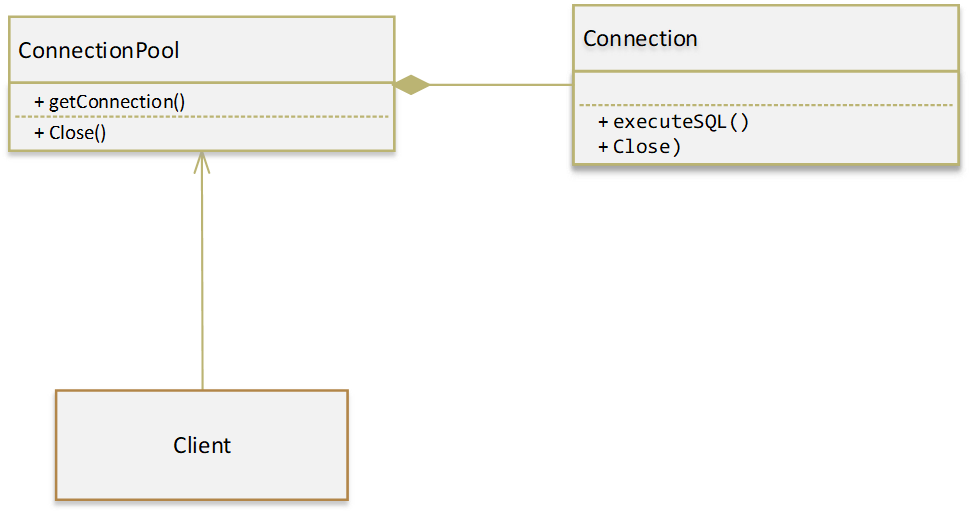
My final schema is to use strategy patterns here. Based on the m, I choose different algorithm to achieve the best effective. By the way, all the algorithm are generics based, so the process is more time consuming.

## 5.5 Factory

****

Create the recommendation algorithm based on the recommendation type: Collaborative Filtering or Co-occurrence.

## 5.6 Flyweight

****

The connection pool would accept the construct parameters, like maximum size, minimum size value and connection string and create connections at once, then available connection would be got by getConnection method and when the close method is called, instead of close it, instead the connection would be returned to the pool.

## 5.7 Template



Another example, since the recommendation system evolves many matrix computing, how about the case that 1 million customers purchase thousands of products? The effort is huge and can’t be able to do within the memory. And in this case, I considered to use map-reduce to compute co-occurrence. I used the map-reduce framework and tried to make it more abstract.

## 5.8 Decorator



JDBC profiler to diagnose the performance of the database access.

## 5.9 Observer



The Observer pattern is used to update the recommendation information. Purchase and Search product category will invoke the updating recommendation information to other registers.

### How to Use

1. **Navigation Framework**

INavigatorFactory

IPageFactory

APageNavigator

INavigatorState

APage

Override createPageA, createPageB and createNavigator methods of INavigatorFactory for page and navigator objects creation.

Override navigate method of APageNavigator and INavigatorState to implement real navigation.

Override open and navigate methods of APage to implement navigation details.

1. **Membership management**

**Implement user information and login’s encode/decode and fileds’ validation.**

ProxyFacade

IEncryption

IHashing

IValidate

ITemplateFacade

Override getHashString, verifyStringByHash, encryptString, decryptString, validateString of IProxyFacade

Override encryptString, decryptString of IEncryption

Override getHashString, verifyStringByHash of IHashing

Override validateString of IValidate

Override SignUp, login, updateProfile of ITemplateFacade

1. **Database connection pool:**

**Three ways to constuct: load from configure file, pass the JDBC connection parameters, pass the JDBC connection parameters and PoolConfigure**

1. **FacadeJDBCExecutor**

**Construct it use default constructor function which would use the JDBC connection pool created by configure file.**

**Then can execute it use following method:**

public boolean executeSQL(String sql)

public T executeSelectSQL(String sql, Function<ResultSet, T> fun)

public List<T> executeSelectSQLAll(String sql, Function<ResultSet, List<T>> funList)

public int executeSelectCount(String sql)

public boolean executeSelectExist(String sql)

public int executeUpdate(String sql)

public int[] executeBatch(List<String> sqlList)

public int executePrepareSQL(String sql, Map<Integer, Object> parameters)

public int executePrepareSQL(String sql, List<Map<Integer, Object>> parameters)

1. **GenericDaoAccess:**

public GenericDaoAccess(String tableName, String keyName)

void insert(T entity) throws SQLException;

void update(T entity) throws SQLException;

T getById(ID id) throws SQLException;

List<T> getAll() throws SQLException;

<R> void executeAll(Functor<T, R> func, Predicate<T> p) throws SQLException;

T findOne(ID id) throws SQLException;

boolean exists(ID id) throws SQLException;

Iterable<T> findAll() throws SQLException;

Iterable<T> findAll(Iterable<ID> ids) throws SQLException;

long count() throws SQLException;

void delete(ID id) throws SQLException;

void delete(T entity) throws SQLException;

void delete(Iterable<? extends T> entities) throws SQLException;

void deleteAll() throws SQLException;

1. **Recommedation System**

**Step 1:**

**RateInformaton assign: the customer numbers, product numbers and rate matrix by override the setUp() method to read input.**

public boolean setUp() :

@Override

**Step 2:** **call predict() to get the target productNo array.**

public Integer[] predict(int product);

1. **Map-reduce framework to compute Co-occurence**

**Step 1: Pass the file Name to the CoOccurence computing class:**

**The file contains the purchase products records.**

CoOccurenceComputing(String fileName)

**Step 2:**

Then Call public void exectue()