DOCUMENTATION

ASSIGNMENT *3:Order Management System*

STUDENT NAME: Robotin George Catalin

GROUP: 30422

# CONTENTS

[DOCUMENTATION 1](#_Toc135245161)

[ASSIGNMENT *3:Order Management System* 1](#_Toc135245162)

[CONTENTS 2](#_Toc135245163)

[1. Assignment Objective 3](#_Toc135245164)

[2. Problem Analysis, Modeling, Scenarios, Use Cases 3](#_Toc135245165)

[3. Design 4](#_Toc135245166)

[4. Implementation 6](#_Toc135245167)

[5. Results 8](#_Toc135245168)

[6. Conclusions 8](#_Toc135245169)

[7. Bibliography 8](#_Toc135245170)

# Assignment Objective

*The main objective of this assignment is to design an orders management application for processing orders of clients, like in a warehouse.*

*Sub-objectives include:*

* *Using a layered architecture*
* *Storing data in a relational database*
* *Using reflection to display and update tables*
* *Using Java records for the Bill class*

# Problem Analysis, Modeling, Scenarios, Use Cases

*Functional requirements for this assignment are derived from the real world scenarios possible when managing a warehouse. i.e. adding/deleting products or clients, storing their orders and generating the corresponding bill.*

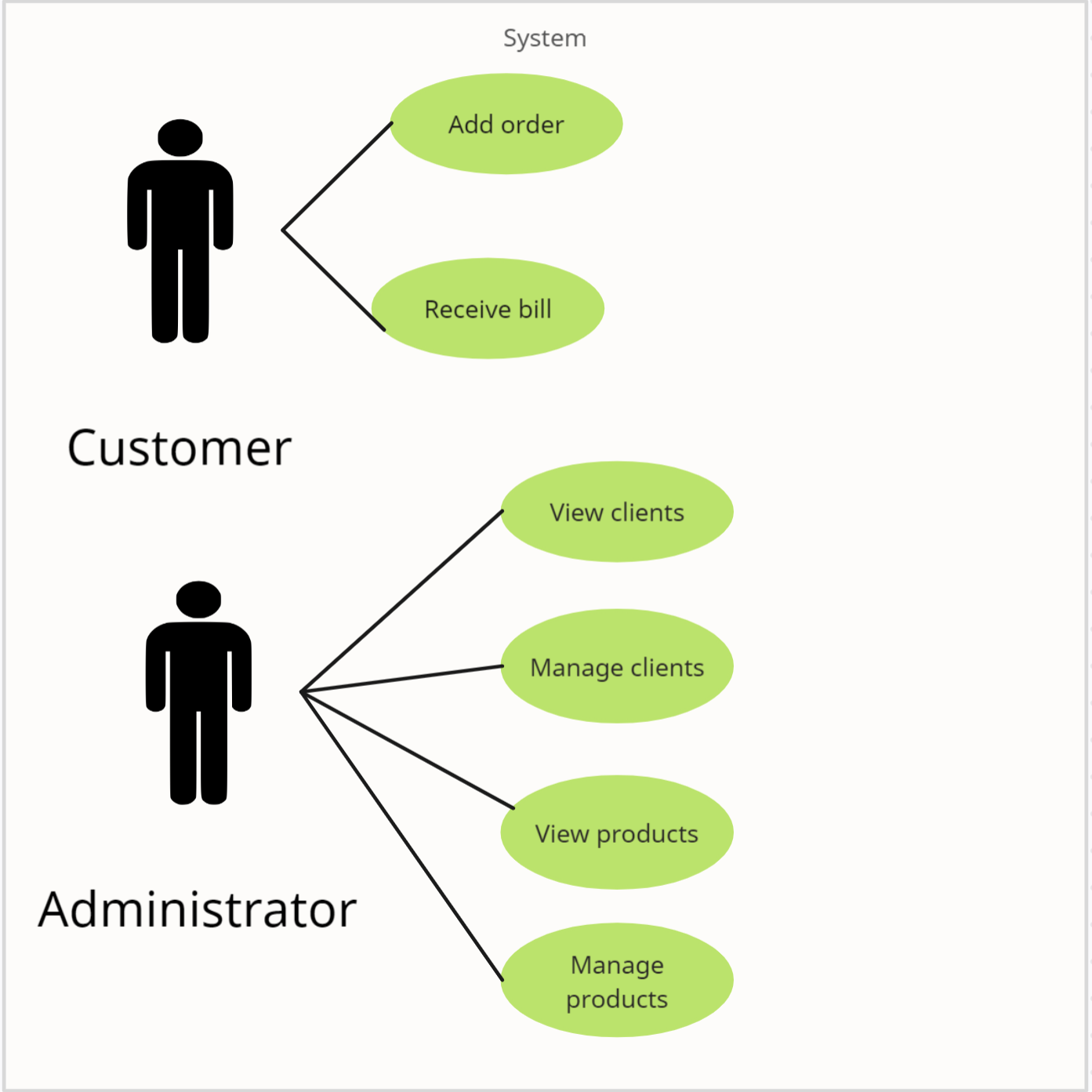


Figure 1:Use case diagram

*For this application, we could consider 2 different actors, namely the user, who just makes orders based on their already known id, and the administrator, who has access to all the products and the clients and can manage them accordingly, i.e., insert, delete, or update information.*

*Successful use case scenario for ordering(client):*

* *Client chooses their id from the drop-down list*
* *They choose the product they want to order*
* *Choose the quantity*
  + *Press the Order button*
  + *A message confirms the order and specifies their total price for the order*

*Unsuccessful use case scenario for ordering(client):*

* *Client chooses their id from the drop-down list*
* *They choose the product they want to order*
* *Choose the quantity*
  + *Press the Order button*
  + *A message popup tells them that the quantity they want to order is bigger than the warehouse current stock*

*Successful use case scenarios for administrator:*

* *Viewing clients or products:*
  + *Choose the corresponding button and press it*
  + *A table appears with all the information about all products/clients*
* *Managing clients/products:*
  + *Press the button from the main menu corresponding to the desired action*
  + *Two windows appear side-by-side: one with the operations possible and one displaying the current clients/products*
  + *The administrator enters the relevant information and presses the corresponding button*
  + *A message pops up, confirming the operation*
  + *The update is seen in real time in the table on the side*

*Unsuccessful use case scenario for administrator:*

* *Managing clients/products:*
  + *Press the button from the main menu corresponding to the desired action*
  + *Two windows appear side-by-side: one with the operations possible and one displaying the current clients/products*
  + *The administrator enters wrong information and presses the corresponding button*
  + *A message pops up, specifying the error*
  + *No update is made*

*Non-functional requirements include:*

* *Having a user-friendly GUI*
* *Protection, by storing information in a database*
* *Reusability, through use of reflection*

# Design

*The application uses an OOP design , namely a layered architecture, having:*

* *A model package, to represent all the data structures needed*
* *A business logic package, used to perform operations on the models*
* *A data access package, used to facilitate communication with the database*
* *A connection package, used to establish connection with the database*

*No interfaces were defined, and no specific algorithms were implemented.*

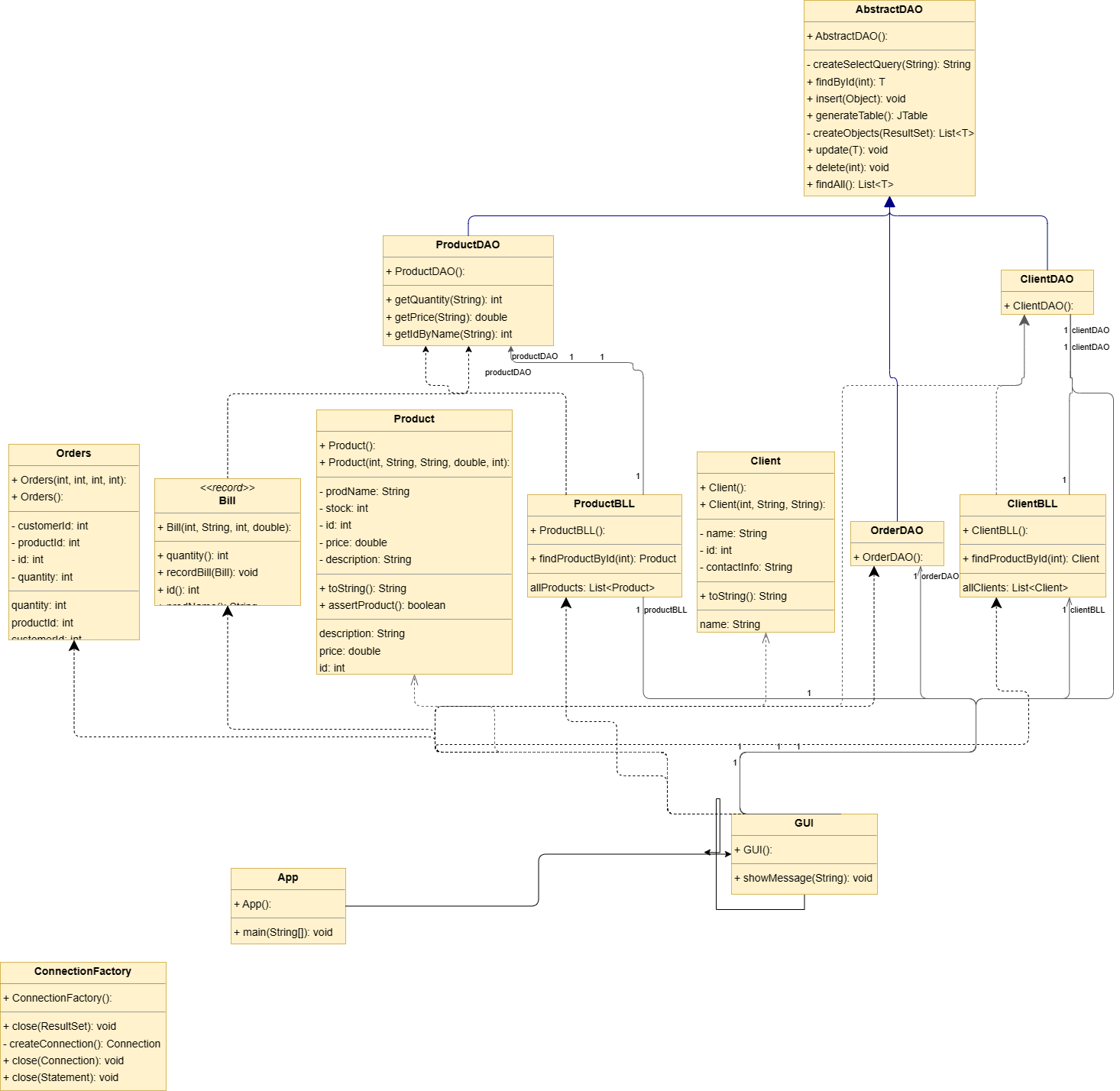


Figure 2:UML Class Diagram

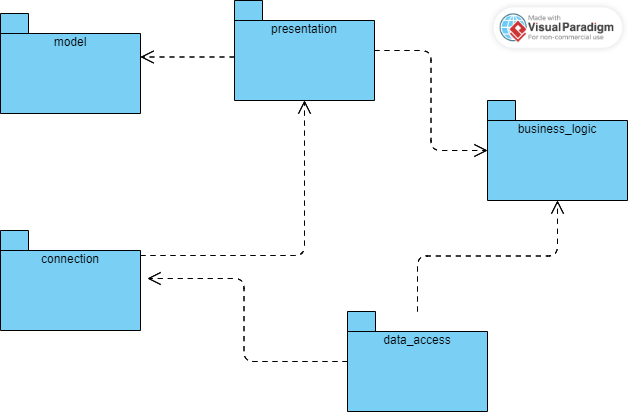


Figure 3:UML Package diagram

# Implementation

*Model package:*

* *Client class*
  + *Id, name, contactInfo: used to describe a client*
* *Product class*
  + *Id, prodName, description, price, stock: used to describe a product*
* *Orders class*
  + *Id, prodId, clientId, quantity: used to describe orders contained in the database*
* *Bill: record: id, prodName, quantity, price*
  + *Static void recordBill(): used to store the record in a database table*

*Connection package*

* *Used to establish connection to the Postgres database*
* *Singleton design: singleInstance object*
  + *getConnection()*
  + *createConnection()*
  + *close()*

*Data\_access package:*

* *AbstractDAO class: uses reflection to generate queries for accessing the database*
  + *List<T> createObjects():creates a list of objects based on the type of the resultSet given at runtime*
  + *List<T> findAll(): returns a list with all entries of a table*
  + *void insert(Object o): inserts an object into the corresponding table*
  + *void update(T t): updates the fields of an entry based on the type of object given*
  + *void delete(int id): deletes an entry of a table based on its id*
  + *T findById(int id): returns an entry of a table based on its id*
  + *JTable generateTable(): generates a Java table from the database one*
* *ClientDAO class: extends AbstractDAO<Client> (no new methods)*
* *OrderDAO class: extends AbstractDAO<Order>*
* *ProductDAO class:*
  + *updateStock(): used to update the stock after an order*
  + *getIdByName(): used to populate order table*
  + *double getPrice(): get price by name; used to populate bill table*

*Business\_logic package:*

* *ClientBLL class:*
* *String[] getIDs(): gets all ids of current customers(uses findAll()), to populate the comboBox from the order table*
* *List<Client> getAllClients(): returns a list of all clients*
* *Client findClientById(): returns a client based on their id*
* *Similar methods in the ProductBLL*

*Presentation package:*

* *GUI class: everything instantiated in its constructor*
  + *showMessage() method: for popup messages after certain actions*

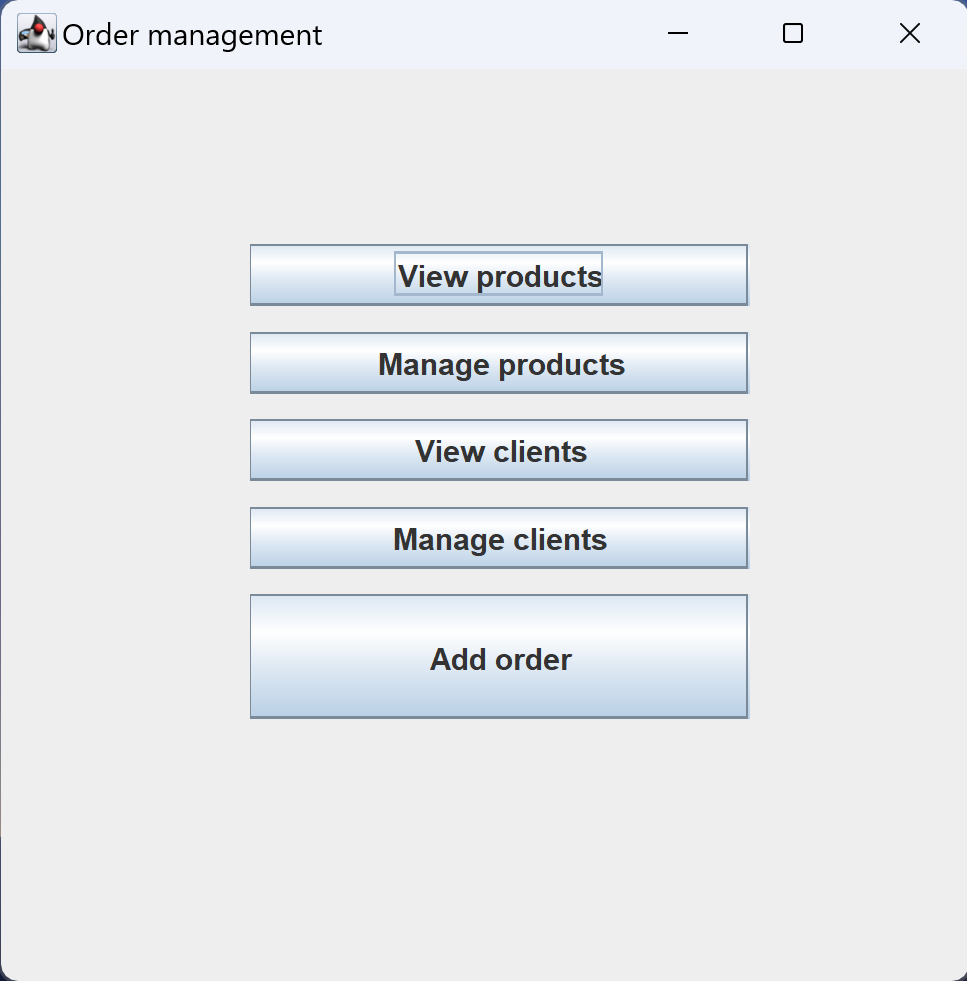


Figure 4:GUI Main view

# Results

*No certain testing scenarios were provided, but through testing while developing the application, all features were functional, from displaying the tables, to updating them or inserting/deleting from them, both in the GUI and the actual database, provided the inputs were correct.*

# Conclusions

*While implementing this application, I learned:*

* *How to use reflection and why it is useful to generate generic methods*
* *How to use Java records*
* *How to work with JTables and database tables*
* *How to generate JavaDoc files*

*Future improvements include:*

* *Having a nicer GUI*
* *More thorough testing of all features*
* *Designing a more detailed database*
* *Adding more functionality, not just inserting orders*

# Bibliography

1. [*https://online.visual-paradigm.com/diagrams/features/package-diagram-software/*](https://online.visual-paradigm.com/diagrams/features/package-diagram-software/) *for package diagram*
2. [*https://app.creately.com/d/7B5TbhBfNFb/edit*](https://app.creately.com/d/7B5TbhBfNFb/edit) *for use case diagram*
3. [*https://gitlab.com/utcn\_dsrl/pt-reflection-example*](https://gitlab.com/utcn_dsrl/pt-reflection-example) *for reflection example*
4. [*exported\_from\_idea.drawio - diagrams.net*](https://app.diagrams.net/?&highlight=0000ff&edit=_blank&layers=1&nav=1&title=exported_from_idea.drawio#R7V3rjtvGFf6fpxA2gLFFstmZIYeXlWVgbTfuFnbtdF20QZAfXHFWYk1xGJKyV32lPEL%2F5ck6F5KiuJRWHFq8qIJha0QezfV8Z85855B%2Bvnh4Eznh%2FB11iT9yHyZn4GzkrsTHLPLcyRmUhVvvP4R94ZeXnkticSOh1E%2B8UH6Z0iAg00SUnSiiX%2BTle%2BrLWkJnRkS9vHA7dXwiLt8500%2BziC4DJhXQgJyNFk4yF4Lx3HHpF1588c1o9DyiNOEFVlw8vCK%2BP%2BL9Y3KXj6%2BK9iISJNsEAjZgfS3E5D%2BTKCEPsuj4S9a7Z34yDkdxsmJdffbbkibjhRPNvOAKhA9p8SKh4ZXOvrKfJheO782CqymrkURj%2BYtnM%2FaX1XPHC9d3cRI50%2BT19Xt%2B7WN28%2FIuL4VZaR6NYj7nshooPy6zu9XdGvE%2FvDe%2BF5CLOfFm8%2BQK%2FmAU%2BvLdqNCJ8z9djVpp9WI0jYiTkFviMxX5aUmi1fltEnnBjPVAFsQkRZeyj%2Fde4L5c3bjnXpAwiY8bN70gZkt1%2Fv7u36wudvcz9dwNgRkJSMRa%2B%2Bjc%2BYSP8a%2BiVJDJ%2BiPriM%2F%2FTuKln9wSXt1bL06eOYtwLFaIF7IlzOpfhi777fnHqrYZjAi7J%2Fv96C4f17Xvn%2B9q5uCLwfQ%2FfrL9s7T2Mw4Lj6H1muv2hKn7WGj5xCf3yZiyu%2Fc%2Bw%2Bi95%2Fvjexokwk5AXZR%2FdBaev5r8hfifCa9jPE8W%2FgSOBdqJOwHjFOFgPKULb8o%2BfeeO%2BC9zk%2FCK%2BjSacLMwZkpLP5F%2Fei4zDnB8JhEtMP2G0AVJotWIgdc28NmIma8LyD6%2FCOEzhBjQ5UywL9hg5omZpln6q7V1uJTmodpWoNZtRRi2bxrCsDWT8B2z816QWoFffi2gZdAKiKBUQAgRzDUQYrDWQEtXUkDYogJ6opAah%2BwzIlMaubmlKFoM79mmicyV%2BCVblda1mDfKDfD36c7y%2FUh8cemSbQLtqfdvSydIvGTFEcU6sLEPyLkUHeX%2FVG0Vnlv1wzCi7t%2BchdjVKvbNMPKm4p4c7ODRBFM0GbAAJhuuwQQNNXNutG3OX%2FkekWvZKhZks5lRz%2FUkvbwBktwfa8kjDJgaVyjxBVujst5f8MNFwnzWm%2BCeFn%2FTwgwmVDa3ibjDO2lbpqdidnZPzkCRbyGJfGwaa%2BRbhW0UaWrIt7pB%2Fsu3bzsCP2u5RaeOn3A%2BsE1qOU0KB7iC9Ts4cBzfl82VDjlpHx4f6EY3r%2BMMN7%2F8egR7pmHLTVNDWzZNTdEFhd1gJ6VLOsDOI45kqDqBkCZ0QtOstU6ggk5gS00lWj8Xv5I0o0eDH9meR6NV%2B6pR7kGL5nXq05hs0FWlg0PGbK07yXu3%2FrbhBsra1jerziFS5jZhdS7IBqt1eFM%2BzXtWHsKg0XhhmZIjQLq%2BZqmgWTzWKMIRtA3HN%2F%2B4aR2ArM0WIRfP6Zd3JI6dGVlz1kfBVZl8M2B6aNtbTteKVJXWthK%2Bj1wSdeElZA0fh5MAkCEdR8MsOAmosZNgdqIPcTfaEEtWZeOfMv2SCrYYfVvGCVuy6KaCWgnlaa3qVhUTk9Ga%2BZ3DuwGPmswOblu7vmPAxUEN24swJVqxUWBITNg81GW3DdeUL2gdr2m7j9jR7HoFPZqFEYqwbgG%2FGe1fyZcyTZ9%2B2g%2B8Ij5QjA7kd1wSTyMvlL52DwjWfDGcmEf7C0t1R6lPnKAd07NlWtYGqHo%2Bq1jaymWqXNlBmyUNycMNLsQ%2FhSXKzJKmq5kl0JFZ6oK5XTfdOXVbNM5tcLdpeyXyNutFBXubOgEcRMdE4xpAe8TiIoCas7i4Ixx1cT5bN90ijmYk%2BSmL%2FeeEQTmQz4Q%2BiGB9LvFoExFCN%2B7LlQj4b61JZqPd8t3lvJDwULVTDhQKOtQFFJCtFZK6tAIU9PpQIO6M8Hr5Z1H9N9AR02U) *for UML class diagram*