Cairo University Faculty of Computers and Artificial Intelligence



**Advanced SE**

**Software design specification document**

**2022**

**Project Team**

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **Email** |
| 20201074 | Rawda Mohammed hussin | rawda14moh@gmail.com |
| 20201136 | Farah Ashraf Wafaa | Farahashraf77788@gmail.com |
| 20201218 | Walaa Soudy Ibrahim | walaasoudy36@gmail.com |
| 20200163 | Haneen Ehdaa Ibrahim | haneen.ehdaa@gmail.com |

Contents

[Instructions[To be removed] 2](#_Toc120811426)

[Class diagram design 2](#_Toc120811427)

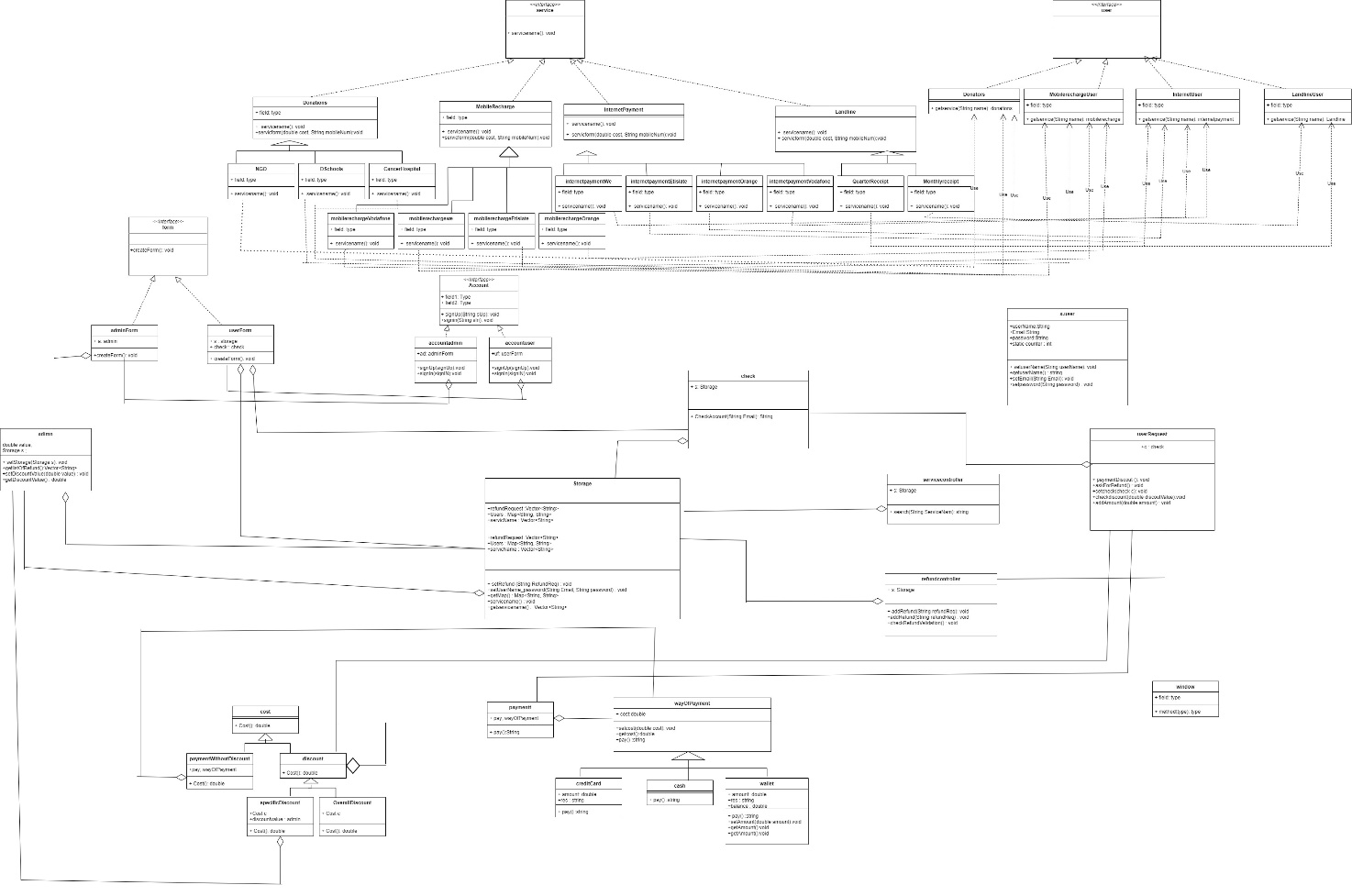
[Class diagram Explanation 3](#_Toc120811428)

[Sequence diagram design 3](#_Toc120811429)

[Github repository link 4](#_Toc120811430)

# Class diagram design

* **You should provide clean version for your class diagram design.**
* **Class diagram is a static diagram and should not represent any dynamic flow of events.**
* **Put Relationships between classes and the types of the relationships.**
* **Put multiplicity.**
* **Put relationship name (e.g. faculty "offer" course).**
* **Put attributes in the classes.**
* **Put functions &Put parameters.**
* **Put data types of each attributes and the parameters.**
* **Highly perfered: Each class has a corresponding interface**
  + **Let all objects parameters and returns be of interface type.**
* **See Shopping Cart Case Study**

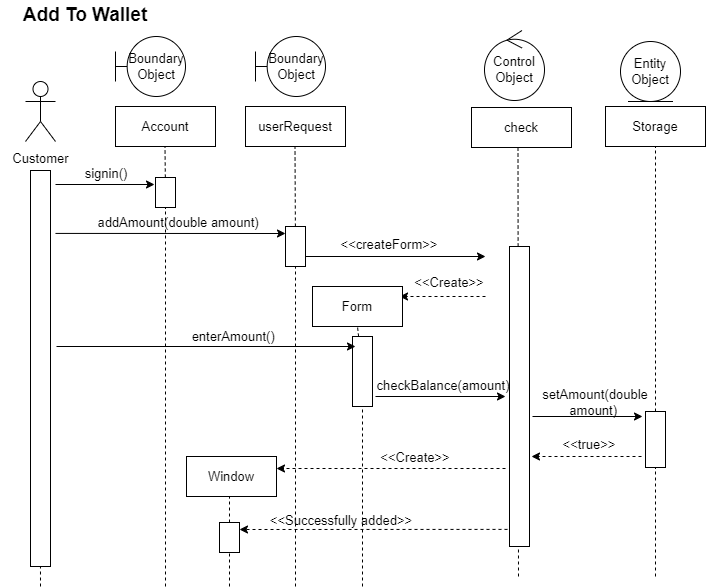


**Note: File of class diagram is attached with this word file for good quality.**

# Class diagram Explanation

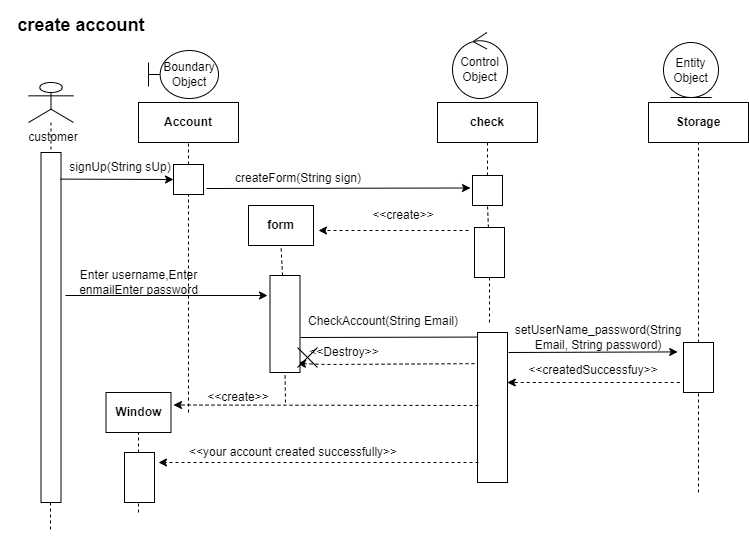
1. (Factory Method pattern): we use factory method pattern in service we put **services** as a product and **user** as a factory creator and the factory method is **getservice.**
2. (Decorator pattern): the decorator is a **discount** and concrete decorator are (**overall discount and specific discount**) and the component is **Cost** and concrete component is **PaymentWithOutDiscount**
3. (Strategy pattern): context **payment** and **wayOfPayment** is a strategy and concrete strategy (**cash, creditCard and wallet**).
4. We apply a single responsibility principle in all our classes.
5. We apply an interface segregation principle :(**Account** is an interface and concrete classes is (**userAccount and adminaccount**) and (**form** is an interface and concrete classes is (**adminform and userform**) and (**user** is an interface and concrete classes is (**mobileRechrageUser , internetPaymentUser, landlineUser, Donators**) and( **service** is an interface and concrete classes is **internetPayment , donations , landline, mobileRecharge**).
6. We apply open closed principle in all our classes.
7. We apply the encapsulation in our classes.

# Sequence diagram design



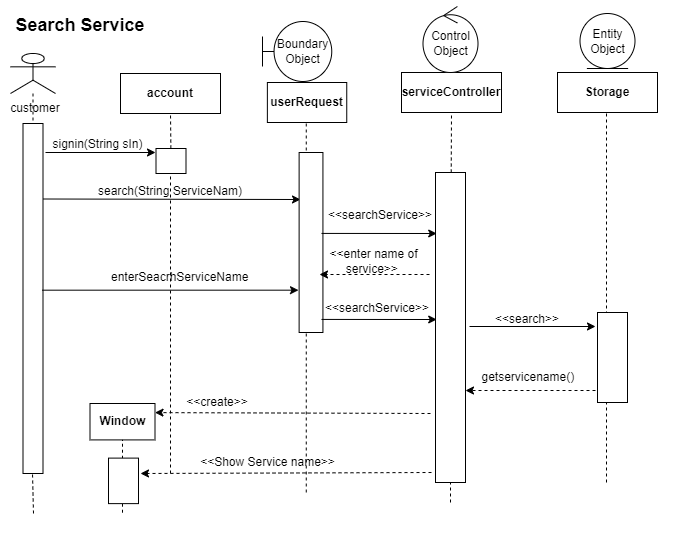
**Add wallet description:**

User login into system and system check login then admin press adds to Amount and then and controller creat form then user fill the form and submit it controller(check) check balance of amount then storage set the amount controller get the state of add in wallet process then controller creat window and send state of add to wallet process

****

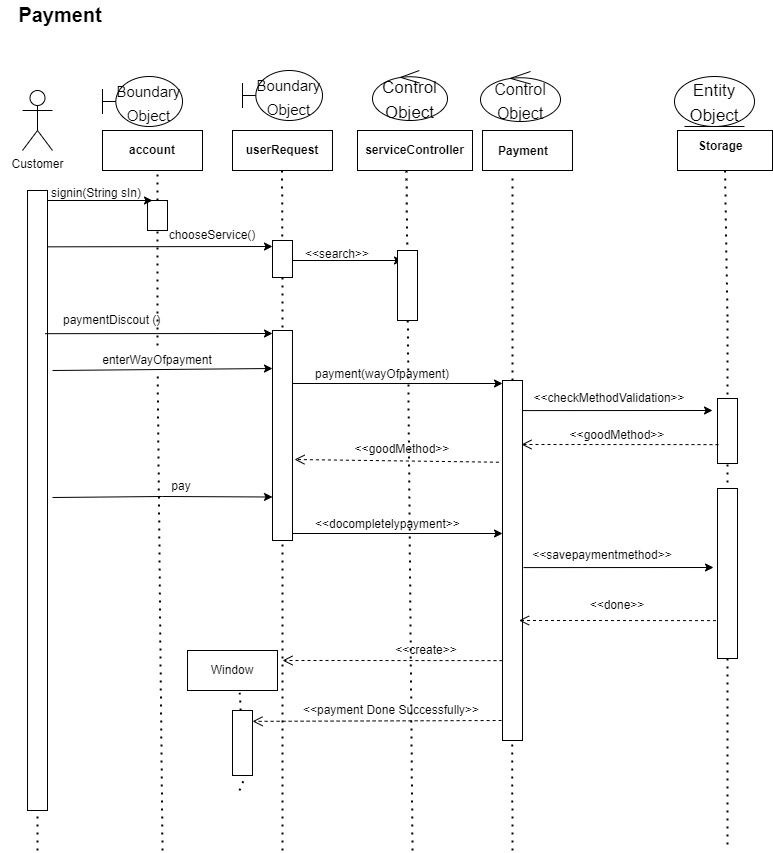
**Create account description:**

The customer chooses to sign up then the form is created, the customer fills it in with his data (password, name, and email) and the controller checks it in the database to see if this account already exists or not then if it does not create this account successfully and display a success message for the user



**Search service description:**

The customer login to the system and then choose search so the user enters the service name then the controller checks, if there is a matching service, exists returns the result, and shows the result to the customer



**Payment description:**

# User will login to fawry system the customer will choose service then userRequest will search if the service chooses by customer is valid then that customer will want to pay then the customer will choose way of payment (controller(payment)) send massege to storage to check if this method is available by check the storage (controller(payment)) will get the state of check allow user to pay then (controller (payment)) will make payment process and get any required data for pay from storage to complete the payment process then send to user window which successful payment.

# 

**Pay with discount description:**

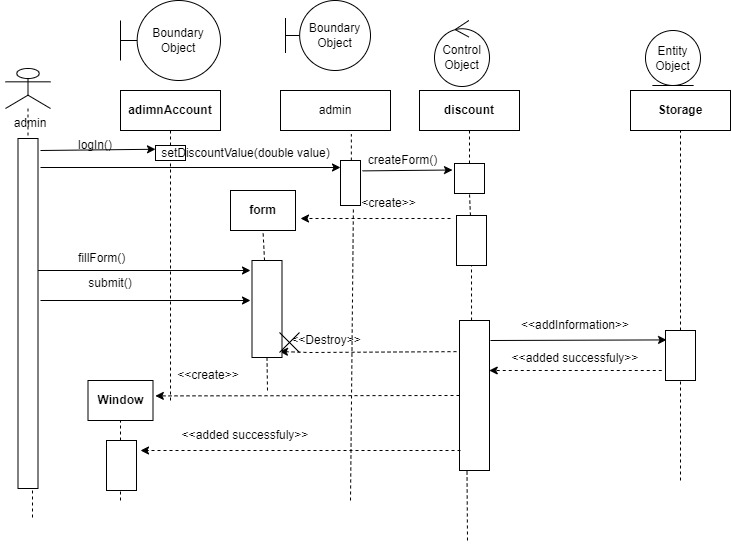
# The customer signs in to the account, then choose to pay with a discount, so the userRequest searches for the services through the services controller, and sends him whether there is a discount or not, then the customer chooses the discount. After that, the userRequest calculates the cost, then sends the service controller a message that he checks the payment method and it appears the customer will see on the screen that he has successfully paid the discount.

# 

**Ask refund description**

Expected that the customer login and choose the service and provider as we put in other sequence diagrams. The customer chooses payment method then controller(payment) put this method into storage and check if it valid then the customer pay button the controller(payment) checks the pay process then storage check payment process and controller(payment) take the state of payment and create a window to show pay state to customer. The customer asks for refund button then controller(refund) create a form to make the customer fills it and submit storage saves the form information controller(refund)check the validation of refund by admin and appear a message to user entity that accepted refund.

Add discount sequence diagram:



**Add discount** **description**

# The admin logs in, then chooses to setdiscountvalue, so a form is created and shown to him, then the admin fills this form with the information he wants, then saves this form, then the controller checks for this information, then it is added to the database, then it appears to the admin that the discount has been added successfully.

**Check refund sequence diagram:**

# 

# Check refund description:

The customer logs in, then asks for a refund, then the controller checks to return the money, then Database stores it, and it appears on the window that it has been checked and added successfully.

# Github repository link

<https://github.com/farahAshraf777/Phase1_software2.git>