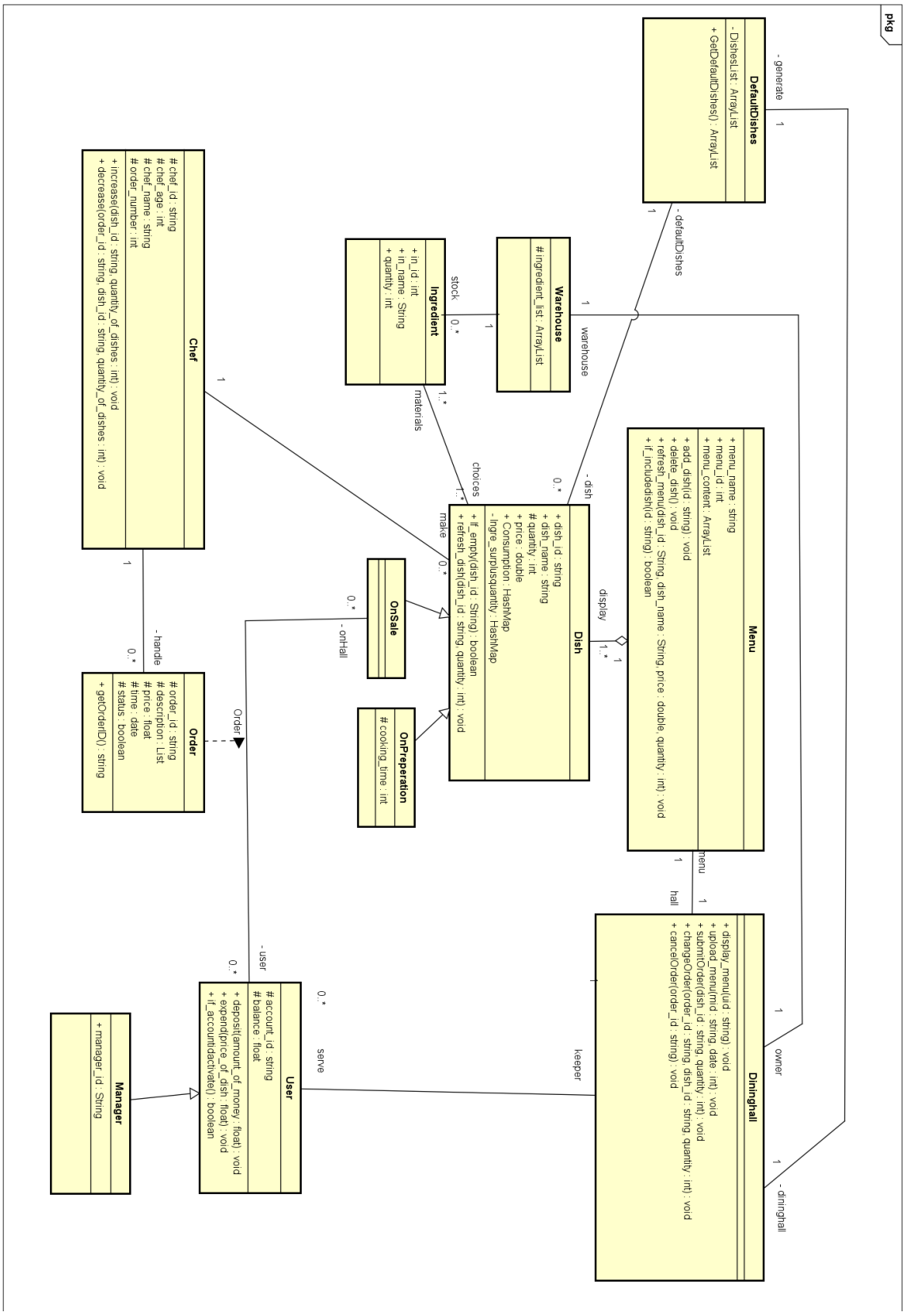
SYSC5709Y
Task 4: Implementation

101198869 Zhuonan Huang
101201163 Di Cao
101198853 Zhixin He
101085230 Mingfang H

1. Updated class diagram



2. Updated use cases

1) UploadMenu

Use case name	UploadMenu
Participating actors	Initialized by Dining hall manager
Flow of events	i) The Dining hall manager logs into her account, clicks the "Upload menu" button and enters today's date in her terminal. ii) After checking the stock of ingredients in the warehouse, Menu responds by generating and presenting default dishes which consist of multiple possible combinations of ingredients. iii) The manager chooses the dishes for today and submits her choice. iv) Menu responds by adding those dishes and forming a menu for today.
Entry condition	• Different available ingredients have already been added into the warehouse. • Dining hall manager has a specific account, and this account has been regarded as the only one to do "UploadMenu" function.
Exit condition	• The manager has received a notification indicating that the menu for today has been formed. • The menu has included all the dishes chosen by Dining hall manager for today.
Quality requirements	• Multiple possible combinations of ingredients for today should be displayed in 30 seconds.

	• The menu should be formed in 10 seconds after the manager submits her choice.

2) DisplayMenu

Use case name	DisplayMenu
Participating actors	Initialized by User
Flow of events	i) The user id is existing. ii) The Menu refreshes. If some dishes are sold out, these dishes will be removed from the menu. iii) The dining hall system responds by displaying the latest menu details to User. iv) The User can scan items in this interface.
Entry condition	• The first menu generation has been completed, which means the "UploadMenu" function has been fulfilled.
Exit condition	• All the available dishes have been displayed in the user interface. • The user chooses to leave the menu interface.
Quality requirements	• The Menu should be presented within 5 seconds after customer application.

3)PrepareOrder

Use case name	PrepareOrder
Participating actors	Initialized by Student Communicates with Chef
Flow of events	i) The Student will create an order when deciding to order something. ii) The Student chooses the Dishes showing on the Menu which are on sale. Afterwards, the Student needs to supplement the information of the Order. iii) The order is finished and submitted to Chef. iv) The chef handles the order and sells the OnSale dishes.
Entry condition	• The Student should log into the dining hall successfully.
Exit condition	• The order is not finished because the Student gives up providing information. • The Student does not pay the money successfully because the money is not enough OR there may be some errors during the process of payment. • The Chef acknowledges that the quantity of Dishes on sale decreases after the order is submitted successfully.
Quality requirements	• The quantity of Dishes should finish changing in 5 seconds after the Student submits the order.

4)PayAccount

Use case name	PayAccount
Participating actors	initiated by students
Flow of events	i) The student taps the card id on the machine ii) If the student activates the 'deposit' function, the amount of balance will increase by the value he inputs. iii) When the order is submitted, there would be a value to pay. The student's account will expend the value of the order.
Entry condition	• The account id should be smaller than 20. Balance should be larger than 0.
Exit condition	• The student's balance will increase by the value they deposit. The student's balance will decrease by the value they expend.
Quality requirements	• The operation that the student does about the balance will be effective in 5 seconds.

3. Updated OCL contracts

(1) The class and invariants

Class name	Invariant	Justification
context Menu	menu_id >= 0	The attribute menu_id is initialized as 0, and after upload_menu(uid, date) operation, it is set as the value of date which is always greater than 0. Thus the invariant of class Menu is menu_id >= 0.
context User	inv: balance>=0 inv: User.allInstances-> forAll(u:User | u<>self implies u.account_id<>self.account_id)	account_id must be unique. It is used for identifying each user. And the balance must be always greater than 0, it could not be overdraft
context Manager	inv: balance>=0 inv: Manager.allInstances-> forAll(m:Manager | m<>self implies m.account_id<>self.account_id) inv: Manager.allInstances-> forAll(m:Manager | m<>self implies m.manager_id<>self.manager_id)	Manager class extends the invariants and methods from User class. Besides, manager should have unique manager_id. It is used for identifying each manager.
context Ingredient	inv: Ingredient.allInstances-> forAll(i:Ingredient | i<>self implies i.in_id<>self.in_id) inv: quantity > 0 inv:in_id> 0	in_id must be unique and greater than 0. It is used for identifying each ingredient. Every ingredient stocked in warehouse must satisfy quantity > 0. Otherwise, it won't exist in warehouse.
context DefaultDishes	true	DefaultDishes is just used as intermedium for manager to choose today's dishes. And can be null if manager doesn't do the upload menu operation. Thus the invariant for this class will always be true.

context Order	inv: Order.allInstances -> forAll(order:Order | order<>self implies order.order_id<>self.order_id) inv: time>=DiningHall.open_hour and time <= DiningHall.close_hour inv: price>=2	Order_id must be unique. It is used for identifying each order. If an instance of order is created, its time should not be null. Each order has the total price for payment. The price should not be negative.
context Chef	Chef.allInstances -> forAll(chef:Chef | chef<>self implies chef.chef_id<>self.chef_id) inv: chef_age>=18 and chef_age<66 inv: chef_name<>null inv: order_number>=0	Chef_id must be unique. It is used for identifying each chef. Chef_age is greater and equal to 18 and less than 66. Each chef should input the name. Each chef should specify the number of orders which is handled, it is beneficial for chef to manage orders.
context Dish	Dish.allInstances-> forAll(dish:Dish | dish<>self implies dish.dish_id<>self.dish_id) inv: dish_name<>null inv: quantity>=0 inv: price>=0	Dish_id must be unique; It can be used for identifying each dish. Each dish should have its name The quantity of dish can be greater and equal to zero. The price of dish can be greater and equal to zero.

context Onsale	Onsale.allInstances -> forAll(dish:Onsale | dish<>self implies dish.dish_id<>self.dish_id) inv: dish_name<>null inv: quantity>=0 inv: price>=0	OnSale class extends the invariants and methods from Dish class
context OnPreparation	OnPreparation.allInstances -> forAll(dish: OnPreparation | dish<>self implies dish.dish_id<>self.dish_id) inv: dish_name<>null inv: quantity>=0 inv: price>=2 inv: cooking_time>=0 and cooking_time <=20	OnSale class extends the invariants and methods from Dish class Cooking time should be greater than zero.

(2) The list of methods in each class

Class Dininghall

signature	display menu()
scope	When customers login in the system, they can view the menu.
Return type	void
type	command
preconditions	self-> menu.menu_id <> null
postconditions	result<>null
justification	Preconditions: (1) The menu_id should be today date. (2) The result of menu should not be null.

signature	upload_menu(mid : string, date : int) : void
scope	This method starts when a manager logs into her account, clicks the “Upload menu” button and enters today’s date in her terminal. This method ends when a menu including all the

	dishes chosen by the manager is formed. During the method process, it invokes DefaultDishes to generate optional default dishes consisting of multiple combinations of ingredients after checking the stock of ingredients in the warehouse, and it is worth nothing that, during this method process, it calculates the maximum quantity that can be cooked of the dish and sets it as the value of attribute quantity of each Dish instance. Then the manager chooses some dishes as today's menu, and system responds by invoking add_dish(id) method recursively to add all the dishes chosen by the manager into the menu. After that, it assigns the value of date to the attribute menu_id to indicate that the menu is for today and has been formed successfully.
Return type	void
type	command
preconditions	date>0 and self.Manager-> exists(m:Manager | m.manager_id = mid) and self.menu.menu_content = null and self.menu.display->isEmpty()
postconditions	self.menu.menu_id = date and self.DefaulDishes.Dish->includesAll (self.menu.display) and and self.menu.display->forAll(d:Dish | self.menu.if_includedish(d.dish_id) and d.quantity > 0)
justification	Before doing this operation, a manager must have valid manager_id to access this operation, the attribute menu_content and the instances of class Dish linked to the instance of class Menu linked to the contextual instance of class Dininghall must be null. After the operation, menu_content and the instances of class Dish linked to the instance of class Menu have been changed and must include all the new added dishes. Besides, the maximum quantity of the added dish is calculated when doing the upload menu operation. Thus, the attribute quantity becomes greater than 0 for each added dish.

signature	DiningHall::submitOrder(dish_id, quantity)
scope	The order is submitted to the chef
Return type	void
type	command

preconditions	self.Menu.OnSale -> exist(onsale:OnSale | onsale.dish_id=dish_id and quantity>0 and quantity<=onsale.quantitiy)
postconditions	self.User.Order.Chef -> exist(chef:Chef | chef.order_number=chef.order_number@pre+1 and self.User.Order -> exist(order:Order | order.order_id=order_id)
justification	User can only choose the dishes which are OnSale. The quantity of each ordered dish should not exceed the OnSale number.After the order is submitted to Chef, the order number should increase. A valid order_id should be existed.

signature	DiningHall:: changeOrder(order_id, dish_id, quantity)
scope	The type of dishes in this order may be changed or the quantity may be changed.
Return type	void
type	command
preconditions	self.User.Order -> exist(order:Order | order.order_id=order_id) and self.Menu.OnSale -> exist(onsale:OnSale | onsale.dish_id=dish_id and quantity>0 and quantity<=onsale.quantitiy)
postconditions	self.Order->exist(order:Order| order.description.size()<>order.description.size()@pre or order.description .szie()=order.description.size()@pre)
justification	The order_id should be existed. The dishes should be OnSale. The quantity should not exceed the OnSale number.After the order is changed, the description of some orders may be changed. This command happens before the order is submitted.

signature	DiningHall::cancelOrder(order_id)
scope	The order is cancelled for some reaosn
Return type	void

type	command
preconditions	self.Order -> exist(order:Order | order.order_id=order_id)
postconditions	self.Order->forAll(order:Order| order.order_id<>order_id)
justification	The order_id should be existed.After the order is deleted, the set of order instances cannot find one whose order id is equal to order_id. This command happens before the order is submitted.

Class Menu

signature	add_dish(id : string) : void
scope	This method starts when there is a request to add a new dish with specific dish_id into the menu. This method ends when the new selected dish is already added into the menu.
Return type	void
type	command
preconditions	id<>null and self.hall.DefaulDishes.Dish->exists(d:Dish | d.dish_id = id and d.quantity>0) and not self.if_includedish(id) and self.display-> not exists(d:Dish | d.dish_id = id)
postconditions	self.if_includedish(id) and self.display->exists(d:Dish | d.dish_id = id)
justification	Before doing the operation, this dish must belong to the optional generated default dishes, but can't exist in the menu. And the attribute quantity has already been calculated and assigned as the maximum quantity, which is always greater than 0. After doing the operation, menu_content must include that new dish's id, the instances of class Dish linked to the instance of class Menu includes that new dish.

signature	delete_dish()
scope	When all this dish is made, remove it from the menu.

Return type	void
type	command
preconditions	self ->exists (Menu:menu | menu.if_includedish(cid) = true) and self.Dish ->exists(Dish:dish | dish.dish_id=cid and dish.if_empty=true)
postconditions	self->exists (Menu:menu | menu.if_includedish(cid) = false)
justification	Preconditions: (1) Dish with this id exists in the menu (2) The quantity of this dish is equal to 0 Postconditions: (1) The dish with this id not exists in the menu

signature	refresh_menu()
scope	After the first version of menu has been uploaded, timed refresh menu. To make sure that everything that can be sold is on the menu and completed dishes are not on the menu.
Return type	void
type	command
preconditions	self->exists(Menu:menu | menu.menu_id=todaydate)
postconditions	self->select(Menu:menu | menu.display.quantity=0)->isEmpty()
justification	Preconditions: (1) The first version of today's menus has been uploaded Postconditions: (1) If all this dish is made, it will disappear from the menu.

signature	if_includedish(id : string) : boolean
scope	This method starts when there is a request to search whether a dish exists in the menu. This method ends when return a query result after searching.
Return type	Boolean
type	Basic query

preconditions	id<>null
postconditions	self.if_includedish(id) implies self.display->exists(d:Dish | d.dish_id = id)
justification	To do this query, we just need to check whether attribute menu_content includes the given id or not. If it includes then returns True, else returns False. And attribute menu_content can be accessed without invoking other query, thus it is a basic query. And the precondition of it is that id is not equal to null.

Class DefaultDishes

signature	GetDefaultDishes():ArrayList
scope	This method starts when there is a request to acquire generated default dishes. This method ends when return the result of all generated default dishes.
Return type	ArrayList
type	Basic query
preconditions	true
postconditions	self .Dish.Ingredient ->forAll(i:Ingredient | Ingredient.quantity >0)
justification	After invoking this method, system will return the ArrayList of generated default dishes. And because these default dishes are generated and composed of available ingredients in the warehouse, the quantity attributes of these dishes' ingredients must always be greater than 0.

Class Dish

signature	If_empty(dish_id)
scope	Used to determine whether the dishes have been sold out.

Return type	Boolean
type	Basic query
preconditions	dish_id<>null
postconditions	result=true =>self.quantity=0
justification	Preconditions: (1) the dish_id should not be null Postconditions: (1) The result is true, which means the quantity of this dish is equal to zero.

signature	refresh_dish(dish_id,quanitiy)
scope	Chef should use this method to make more OnPreparation dishes. It also increase the quantity of dishes.
Return type	void
type	command
preconditions	dish_id<>null and OnPreparation.quantity=0
postconditions	OnPreprartion.quantity=OnPreparation_quantity@pre+quanitiy and Dish.quantitiy=Dish.quantity@pre+quanitiy
justification	The command needs a valid OnPerparation dish_id and quantity. Onpreparation_quantitiy should be zero. The number of OnPerparation dishes increases. At the same time, the number of total dishes increases.

Class User

signature	deposit(amount_of_money : float)
scope	this function starts when user activates the deposit function, ends when the amount_of_money has been added in the account

Return type	void
type	command
preconditions	amount_of_money>0 and if_accountidactivate()
postconditions	self.balance=self.balance@pre+amount_of_money
justification	By doing this command, we could calculate the amount_of money the student has after deposit money. The amount is increased by the amount of money student deposit.

signature	expend(price_of_dish:float)
scope	this method starts when the user activates the expend function, and ends with price_of_dish has been subtracted from the account
Return type	void
type	command
preconditions	self.balance>self.order.price and self.order->exist(order:order|order.order_id=oid , price_of_dish=self.order.price and if_accountidactivate()
postconditions	self.balance=self.balance@pre-price_of_dish
justification	this method could ensure that the student's balance in the account is after subtracting the price of his order. The student has submitted the order, the balance in the account is larger or equal to the amount of the price of order,the price_of_dish is equal to the price of order. The amount in student account is equal to the @pre balance subtract the price of the dish.

signature	if_accountidactivate()
scope	it starts by the operation is used in the account.it ends when there is no operation in the account
Return type	Boolean
type	Basic query
preconditions	Account_id<=20

postconditions	result=true =>account_id<>null and account_id<=20
justification	this could help us know whether the account could be used in operation

Class Order

signature	getOrderID()
scope	Acquire orderID and return
Return type	order_id:string
type	Basic query
preconditions	
postconditions	result=order_id
justification	A valid order id can be acquired. This is a basic query; it cannot change the status of Order. There is no postcondition

Class Chef

signature	increase(dish_id,quantity_of_dishes)
scope	Chef needs to move OnPreperation dishes to OnSale
Return type	void
type	command
preconditions	self.OnSale -> exist(onsale:OnSale | onsale. quantity=0) and self.OnPerparation -> exist(onpreparation:OnPreparation | onpreparation.quantity>=0)
postconditio	self.Onsale -> exist(onsale:OnSale | onsale.quantitiy=onsale.quanitiy@pre+quantity_of_dishes)

ns	and self.OnPerparation -> exist(onpreparation:OnPreparation | onpreparation.quantitiy= onpreparation.quanitiy@pre+quantity_of_dishes) and self.Dish->forAll(dish:Dish,ingredient:Dish.Ingredient|ingredient. quanitiy = ingredient.quantity@pre -dish.consumption*quanitity_of_dishes)
justification	The command will execute if there is no OnSale dishes and the OnPreparation dishes is not zero. The number of OnSale dishes increases. The number of OnPerparation dishes decreases

signature	decrease(order_id,dish_id,quantity_of_dishes)
scope	Chef needs to check the quantity of OnSale dishes. If the quantity is enough for sale, chef will handle the order and puts the OnSale dishes to the student. Afterwards, Chef should delete this order.
Return type	void
type	command
preconditions	self.Order -> exist(order:Order | order.order_id=order_id | and order.onHall.quantity<=self.OnSale.quantity)
postconditions	self.order_number=self.order_number@pre-1 and self.OnSale -> exist(onsale:OnSale,order:self.Order | onsale.quantity=onsale.quantity@pre-order.onHall.quantity) and self.Dish -> exist(dish:Dish,order:self.Order | dish.quantity=dish.quantity@pre-order.onHall.quantity) and self.Order -> forAll(order:Order | order.order_id<>order_id)
justification	The order_id should be existed. The quantity of dishes in this order should not exceed the number of OnSale dishes. After the order is handled, the number of orders should decline. The number of OnSale dishes should decrease. The quantity of dishes should decrease. The order should be deleted and the set of order instances cannot have the id of this order.

4. Change document

Date	change id	Description	Justification	Comments
3.16	001	Add a new class called DefaultDishes	This class is used to save all the default dishes for the dining hall.	Generate all default dishes after checking the ingredients in the warehouse
3.17	002	Delete some variables	Delete the variable called expiration_date in class Ingredient.	Since we assume that all the ingredients stocked in warehouse are available and up-to-date.
3.18	003	Add some variables	Add new variables manager_id in class Manager, Consumption and Ingre_surplusquantity in class Dish.	Add manager_id to distinguish manager from common user, so that we can use manager_id to validate manager's unique function like upload menu. Consumption and Ingre_surplusquantity respectively represent the amount of ingredient consumption per dish per portion and the surplus quantity of ingredient if one portion is sold. By doing this, we can dynamically check whether the dish has available ingredients even if it uses the same kind of ingredient as many other dishes.
3.19	004	Change the attributes of some variables	Change the in_id of class Ingedient from String to int.	To simplify and solve some implementation problem when connecting the MYSQL database.
3.20	005	Add the contract of DefaulDishes	Add postcondition : self . Dish.Ingredient forAll i:Ingredient | Ingredient.quantity >0)	The generated default dishes should consist of available ingredients, which means the quantity variable of these

				ingredients should be greater than 0.
3.21	006	Change the contract of add_dish ()	Change precondition: d.quantity = 0 to d.quantity > 0 self.hall.warehouse.stock. choices-> exists(d:Dish | d.dish_id = id) to self.hall.DefaulDishes.Dis h->exists (d:Dish | d.dish_id =id)	We move the dish quantity calculation step out of add_dish() function, now it's done before invoking add_dish(), so that manager can see the available quantity of all default dishes and choose the dishes he wants for today. And since we add a new class DefaulDishes Some navigation expressions can be modified by using the navigation DefaulDishes.
3.22	007	Change the contract of upload_menu mid , date)	● Delete postcondition : self.menu.display.mat erials 20sset forAll i:Ingredient | i.expiration_date > date) ● Change postcondition: self.warehouse.stock. choices includesAll (self.menu.display) to self.DefaulDishes.Dis h . includesAll (self.menu.display)	We assume that the ingredients in warehouse are up-to-date, so there is no need to check date now. And since we add a new class DefaulDishes Some navigation expressions can be modified by using the navigation DefaulDishes.
3.23	008	Change Invariants of manager class	Change the invariant for manager guaranteeing manager_id is not null and is unique.	It is used for identifying each manager.
3.24	009	Change the class diagram of delete_dish	In actual operation, we use the loop to find dishes which quantity is equal to zero, then delete them.	We do not need to know dish_id
3.25	010	Change the class diagram of	Use hashmap to know the quantity of remaining	We need to know attributes when we use

		refresh_menu function	ingredient and the amount of ingredients needed to make a dish. Divide the remaining quantity of ingredient by how much ingredient needed for each dish.	the method. So it changes to refresh_menu (String dish_id , String dish_name,double price, int quantity)
3.26	011	Change the contract of if_empty (dish_id)	Precondition: dish_id <>null Postcondition: result=true =>self.quantity=0	When the result is true, which means the quantity of that dish is equal to zero.
3.27	012	Change the contract of display_menu()	Precondition: self-> menu.menu_id <> null Postcondition: result<>null	Suppose the date entered is today's date.The result should not be null.
3.28	013	Change the invariant of dininghall	Add open time and closed time, the closed time should be bigger than open time.	The order in dining hall should be in opening hours.
3.29	014	Change the contract of changeOrder(order_id, dish_id, quantity)	In postcondition, add a constraint that the size of description should not be changed if no parameters are sent.	We need to consider every situation of description status.
3.30	015	Change the contract of increase(dish_id,quantity_of_dishes)	in postcondition, the chef should ensure that ingredients' number should decrease after dishes are put into OnSale.	We add this constraint to control the quantity of dishes synchronically.
3.31	016	Change the invariant of order class	In variant time. The time of creating order should be in dining hall opening hours. In description, the size should have upper limit.	Because dining hall do not have order without date, the description of order need to have a size.
4.1	017	Change the invariant of dish class	● Add a HashMap Consumption for listing all ingredients of one dish ● Add a HashMap	● The quantity of dishes should be closely related to ingredients, we need to manage the

			Ingre_surplusquantity for listing remaining ingredients of one dish ● In Ingre_surplusquantity, add a constraint that if one dish cannot have at least one ingredients, it will not show on the menu. ● In price, add a specific limit. ● In OnPreparation class, the cookingtime needs a ipper limit	quantity of ingredients ● Price of dishes should have a limitation. ● Dishes in preparation should have limitation in cooking time.
4.2	018	Change the invariant of user class	account_id not to be empty, move it to the if_accountactive() fuction which explain the same thing	As if_acountactive() has judged whether the account id is in the system.
4.3	019	Change the precondition for if_accountactivate () function	I require the number of account_id is not larger than 20.	We assume that we test for 20 users.
4.4	020	Change the postcondition for if_accountactivate () function	If the account_id is larger than 20, it will return false.	It is a Boolean function, if the value of account id is larger than 20, that means it is not in the system, so it returns to false.

5. JML and documentation

if_includedish(id : string) : boolean JML:

```
blic class Menu {

  /*@
    @ public invariant menu_id > 0;
    @*/
  public String menu_name;
  public int menu_id;
  public /*@ nullable @*/ List<Dish> menu_content;

  public Menu()
  {

  }
  /**
   * This method is used to check if a dish already exists in the menu.
   * This method starts when there is a request to search whether a dish exists in the menu
   * This method ends when return a query result after searching.
   * @param id
   * @return Boolean
   */
  /*@
    @ requires id != null;
    @ ensures \result== (\exists int i; 0 <= i && i<menu_content.size(); menu_content.get(i).dish_id == id);
   @*/
  public /*@ pure @*/ Boolean if_includedish(String id)
```

Add_dish JML:

```
  /**
   * This method is used to add the dish chosen by manager into the menu.
   * This method starts when there is a request to add a new dish with specific dish_id into the menu.
   * This method ends when the new selected dish is already added into the menu.
   * @param id
   * @return void
   */
  /*@
    @ requires id != null;
    @ requires (\exists int i; 0 <= i && i<DefaultDishes.DishesList.size();
    @   DefaultDishes.DishesList.get(i).dish_id == id && DefaultDishes.DishesList.get(i).quantity > 0);
    @ requires !(if_includedish(id));
    @ ensures if_includedish(id);
   @*/
  public void add_dish(String id) throws Exception
```

GetDefaultdishes JML:

```
}
/**
 * This function is used to get the generated default dishes from the available ingredients
 * This method starts when there is a request to acquire generated default dishes.
 * This method ends when return the result of all generated default dishes.
 * @return ArrayList<Dish>
 */
/*@
  @ requires true;
  @ ensures (\forall int i; 0 <= i && i<DishesList.size(); (\forall int j;
  @    0 <= j && j< Arrays.asList(DishesList.get(i).Ingre_surplusquantity.keySet().toArray()).size();
  @    DishesList.get(i).Ingre_surplusquantity.get(Arrays.asList(DishesList.get(i).Ingre_surplusquantity.keySet().toArray()).get(j)) > 0));
 @*/
public static ArrayList<Dish> GetDefaultDishes() {
```

Upload_menu JML:

```
  public /*@ nullable @*/ Menu menu = new Menu();
  public List<Ingredient> products;
  /**
   * The upload menu function is used by the manager to choose dishes for today and generate today's menu.
   * This method starts when a manager logs into her account, clicks the "Upload menu" button and enters today's date in her terminal.
   * This method ends when a menu including all the dishes chosen by the manager is formed
   * @param mid
   * @param date
   * @return void
   */
  /*@
    @ requires date > 0;
    @ requires (\exists Manager m; m.manager_id == mid);
    @ requires menu.menu_content == null;
    @ ensures menu.menu_id == date;
    @ ensures DefaultDishes.DishesList.containsAll(menu.menu_content);
    @ ensures (\forall int i; 0 <= i && i<menu.menu_content.size();
    @    menu.if_includedish(menu.menu_content.get(i).dish_id) && menu.menu_content.get(i).quantity > 0);
   @*/
  public void upload_menu(String mid, int date)
  {
```

Manager invariant:

```
public class Manager extends JFrame implements ActionListener{
    /*@
      @ public invariant manager_id != null;
      @ public invariant (\forall int i; 0 <= i && i<m_dininghall.ManInfo.size();
      @   (\forall int j; 0 <= j && j< m_dininghall.ManInfo.size();
      @       m_dininghall.ManInfo.get(j) != m_dininghall.ManInfo.get(i)
      @           ==> m_dininghall.ManInfo.get(j).manager_id != m_dininghall.ManInfo.get(i).manager_id));
      @*/
```

Ingredient invariant:

```
public class Ingredient {
    /*@
      @ public invariant in_id > 0;
      @ public invariant quantity > 0;
      @*/
    public int in_id;
    public String in_name;
    public int quantity;
```

display_menu() JML:

```
/**
 * for the display_menu() method, it used to extract data in database and display the whole menu to customers. After run this method, it will return
 * @param mid
 * @param date
 */
// display_menu() JML
/*@
  @ requires menu.menu_id != null;
  @ ensures menu != null;
  @*/
```

dish invariant:

```
/**
 * Dish_id must be unique; It can be used for identifying each dish.Each dish should have its name.The quantity of dish can be great
 * @param dish_id
 * @param dish_name
 * @param price
 * @param quantity
 */
//dish invariant
/*@
  @ public invariant (\exists Integer i; Ingre_surplusquantity.get(i)==0
  @                     ==> \forall OnPreparation onpreparation; onpreparation.dish_id!=dish_id);//peoblems and errors
  @ public invariant (price>0 && price>=2);
  @ public invariant (\forall int i; 0 <= i && i <display.size();
  @             (\forall int j; 0 <= j && j <display.size();
  @                !(display.get(i).dish_id.equals(display.get(j).dish_id)) && display.get(i).dish_id !=null &&
  @                            display.get(j).dish_id != null));
  @
  @ public invariant (\forall int i; 0 <= i && i <display.size();
  @                     (display.get(i).dish_name != null && display.get(i).quantity >= 0));
  @*/
```

If_empty(dish_id) JML:

```
**
 * if_empty method is used to check if customers can order more this dish. It should use dish_id to check and it will return boolean t
 * @param dish_id
 * @return
 */
//If_empty(dish_id) JML
*@
 @ requires dish_id != null;
 @ ensures \result == (\exists int i; 0 <= i && i<display.size(); display.get(i).dish_id == dish_id && display.get(i).quantity == 0);
 @*/
```

delete_dish() JML:

```
/**
 * delete_dish method is used to delete the dish which quantity is equal to zero. It will loop all dishes in the menu and to check its quantity. It
 * @return
 */
//delete_dish() JML
/*@
  @ requires menu.if_includedish(delNumber) == true;
  @ requires if_empty(delNumber) == true;
  @ ensures menu.if_includedish(delNumber) == false;
  @*/
```

refresh_menu(String dish_id, String dish_name,
String price, String quantity) JML:

```
/**
* When the chef cooks any dishes,refresh_menu method is used to refresh the quantity of all dishes on the menu.It will use the dish_ic
* @param dish_id
* @param dish_name
* @param price
* @param quantity
* @return
*/
/refresh_menu() JML
/*@
  @ requires menu.menu_id == date;
  @ ensures  !(\exists int i; 0 <= i && i < display.size();
  @                  (display.get(i).quantity == 0));
  @*/
```

Chef decrease JML:

```
/**
 * The chef also need to sell the dishes. it should decrese the number of OnSale.
 * @param order_id
 * @param quantity_of_dishes
 * @return
 */
/*@
        @ requires(\exists Order order; order.order_id==order_id && order.status==true)
        @       &&(\exists OnSale onhall, onsale; onhall.quantity<=onsale.quantity);
        @ ensures(order_number==\old(order_number)-1)
        @       &&(\exists OnSale onhall,onsale; onsale.quantity==\old(onsale.quantity)-onhall.quantity);
@*/
```

Chef increase JML:

```
/**
 * The chef need to cook the dishes and put them on sale.
 * so the chef need to move the dishes from onPreparaton to OnSale
 * when the dish is finished, the number of corresponding ingredients also decrease
 * @param dish_id
 * @param quantity_of_dishes
 * @return
 */
/*@
    @ requires(\exists OnSale onsale; onsale.quantity==0)
    @       &&(\exists OnPreparation onpreparation; onpreparation.quantity>0);
    @ ensures (\exists OnSale onsale; onsale.quantity==\old(onsale.quantity)+quantity_of_dishes)
    @       &&(\exists OnPreparation onpreparation;
    @           onpreparation.quantity==\old(onpreparation.quantity)-quantity_of_dishes
    @           && (\forall Integer i; i>=0 && i<onpreparation.Ingre_surplusquantity.keySet().size();
    @               onpreparation.Ingre_surplusquantity.get(i)==\old(onpreparation.Ingre_surplusquantity.get(i))
    @               -(onpreparation.Consumption.get(i)*quantity_of_dishes)));
@*/
```

Chef invariants:

```
/**
 * the chef class describe the main inforamtion and action which
 * a chef need to have.
 * @param chef_id
 * @param chef_age
 * @param chef_name
 * @param order_number
 */
/*@
    @protected invariant (\forall Chef a,b; a!=b ==> a.chef_id!=b.chef_id);
    @protected invariant (chef_age>=18 && chef_age<=66);
    @protected invariant (chef_name!=null);
    @public invariant (order_number>=0);
@*/
```

Dininghall change order JML:

```
/*@
      @ requires (\exists Order user_order; user_order.order_id==order_id)
      @         && (\exists OnSale menu_onsale; menu_onsale.dish_id==dish_id
  @          && quantity>0
  @          && quantity<= menu_onsale.quantity);
      @ ensures (\exists Order user_order; user_order.description.size()
      @          !=\old(user_order.description.size())
      @          || user_order.description.size()==\old(user_order.description.size()));
@*/
```

Dinghall create order JML:

```
/*@
  @ requires (\exists OnSale menu_onsale; menu_onsale.dish_id==dish_id
  @           && quantity>0
  @           && quantity< menu_onsale.quantity);
  @ ensures (\exists Chef user_order_chef; user_order_chef.order_number
  @           == \old(user_order_chef.order_number)+1)
  @           && (\exists Order user_order; user_order.order_id==order_temp.order_id);
@*/
```

Dinghall cancel order JML:

```
/**
 * the Use is allowed to cancel the order if they want to. the order_id should be deleted.
 * @param order_id
 * @return
 */
/*@
        @requires (\exists Order user_order; user_order.order_id==order_id);
        @ensures (\forall Order user_order; user_order.order_id!=order_id);
@*/
```

Dinghall change order JML:

```
/**
 * User can change thge dishes in the order.the dishes must be on sale and have enough numbers.
 * @param order_id
 * @param dish_id
 * @param quantity
 * @return
 */
/*@
      @ requires (\exists Order user_order; user_order.order_id==order_id)
      @         && (\exists OnSale menu_onsale; menu_onsale.dish_id==dish_id
  @          && quantity>0
  @          && quantity<= menu_onsale.quantity);
      @ ensures (\exists Order user_order; user_order.description.size()
      @          !=\old(user_order.description.size())
      @          || user_order.description.size()==\old(user_order.description.size()));
@*/
```

Dininghall create order JML:

```
/**
 * Create the order, user can only choose the order on the menu. It can generate an order ID automatically.
 * @param dish_id
 * @param quantity
 * @return order_id
 */

/*@
  @ requires (\exists OnSale menu_onsale; menu_onsale.dish_id==dish_id
  @           && quantity>0
  @           && quantity< menu_onsale.quantity);
  @ ensures (\exists Chef user_order_chef; user_order_chef.order_number
  @           == \old(user_order_chef.order_number)+1)
  @           && (\exists Order user_order; user_order.order_id==order_temp.order_id);
@*/
```

Dininghall invariant:

```
/**
 * The dining hall should have the open time and close time to spicify
 * thr working hours
 * @param opentime
 * @param closedtime
 */
/*@
    @ public invariant (opentime!=null);
    @ public invariant (closedtime!=null && closedtime.compareTo(opentime)>0);
 @*/
```

Expend JML:

```
/**
 *
 * @param conn
 * @return ar
 */
/*@
 @ requires account_id!=null && pricetotal<=balance ;
 @ ensures balance==\old(balance)-pricetotal;
 */
```

Order invariant:

```
/**
 * This class describe some important information of order
 * @param order_id
 * @param descroption
 * @param price
 * @param time
 * @param status
 */
/*@
  @ public invariant (\forall int i; i>=0&&i<orderlist.size();
  @                     (\forall int j;j>=0&&j<orderlist.size();
  @                         (orderlist.get(i)!=orderlist.get(j))
  @                         ==>orderlist.get(i).order_id!=orderlist.get(j).order_id));
  @ public invariant (description.size()>0 && description.size()<100);
  @ public invariant (\exists DiningHall dn; time.toString().equals("null")==false
  @                     && time.compareTo(dn.opentime)>0
  @                     && time.compareTo(dn.closedtime)<0);
  @ protected invariant (price>=5 && price<=20);
@*/
```

6. Conclusion:

Through implementing design by contract on our project, we used class diagram, OCL, JML to help us have a clear mind to build our dining hall system. During this process, we improved our contract to make our system to be more realistic.
JML help us to describe what the code do precisely. We need to use JML to describe the precondition, postcondition. By following the precondition and postcondition, when we wrote the function, we knew clearly what condition we should satisfy, and what value we should return for this function.

However, when we run the JML. There is an error related to color mistake. The screenshot of this error is attached below. It shows that something wrong with color: “Eclipse.app/Contents/Eclipse/configuration/org.eclipse.osgi/965/0/.cp/specs/java/awt/Color.jml:43: error: The specification must include all the annotations that the Java declaration declares: @java.beans.ConstructorProperties({"red", "green", "blue", "alpha"})public Color(int Param0, int Param1, int Param2, int Param3);

```
[0.00] Executing openjml on Dininghall.java
/Users/zhuonanhuang/Desktop/Eclipse.app/Contents/Eclipse/configuration/org.eclipse.osgi/965/0/.cp/specs/java/awt/Color.jml:43: error: The specifica
    public Color(int Param0, int Param1, int Param2, int Param3);
    ^
```

And we found that this problem might be related to the compatibility problem between OpenJML and JFrame. Since when we didn’t use JFrame, this error disappeared. (The screenshots below show the result). But considering the fact that we need an user interface to do some operations, we decide to keep the JFrame design in code. And we think this is an interesting point for us to do some further research and to see if there is a better solution to solve this kind of question.
The result with JFrame:

```
public class Manager extends JFrame implements ActionListener{
    /*@
      @ public invariant manager_id != null;

[0.00] Executing openjml on Manager.java
Continuing bravely despite parsing errors
/Users/zhuonanhuang/Desktop/Eclipse.app/Contents/Eclipse/configuration/org.eclipse.osgi/965/0/.cp/specs/java/awt/Color.jml:43: error: The specifi
    public Color(int Param0, int Param1, int Param2, int Param3);
    ^
/Users/zhuonanhuang/Documents/workspace/dininghall2/src/com/station/dao/SqlTest.java uses unchecked or unsafe operations.
Recompile with -Xlint:unchecked for details.
[1.03] Completed with errors
```

The result without JFrame:

```
import cn.itcast.frallscore.domain.*;
@SuppressWarnings("serial")
public class Manager extends AccountId implements ActionListener{

    /*@

[0.00] Executing openjml on Manager.java
Some input files use or override a deprecated API.
Recompile with -Xlint:deprecation for details.
/Users/zhuonanhuang/Desktop/dininghall4/src/cn/stcast/warehouse/dao/SqlTest.java uses unchecked or unsafe operations.
Recompile with -Xlint:unchecked for details.
[1.32] Completed
```

By building the dining hall system, we also got the skills in using OpenJML. OpenJML help us to judge whether our JML is legal or not. It helps us to improve the JML accuracy.