Departament d'Enginyeria informàtica i Matemàtiques

Online Restaurant

(Programació & Projectes de Sistemes Informàtics)

NOM DE L'EQUIP: Group 6
DIRECTOR: Oriol Mauri Guiu
Raúl Casanova Marqués
EQUIP: Marc Cabré Guinovart
Albert Cañellas Solé
Bernat Bosca Candel
Aleix Mariné Tena
ASSIGNATURA: Programació
DATA: 13/ 12/ 2016

Contend

Introduction

The project consists to do an application to manage the data from a restaurant.
The application should be able to receive orders, managing them with the correct restaurant and take care of the distribution.

We are a group of 6 people working in this project:

-Aleix Marine Tena (prog4)
-Marc Cabré Guinovart (prog1)
-Bernat Bosca Candel (prog3)
-Albert Cañellas Solé (prog2)
-Oriol Mauri Guiu (secretari)
-Raúl Casanova Marqués (director)

We have divided the code in different packages:

1. The controllers: Classes that manages the data. It has different methods, like the removes, the adds, etc. (ApplicationController, ClientController, OrderController, OrderLineController, ProductController).
2. The exceptions: all the exceptions that the different classes must control in order to don't have any problem or error while executing the application.
3. Models: the "basic" classes like products, client, order. The different classes have their own variables and getters. (Client, Dish, Drink, Order, OrderLine, Product).
4. Utils: Support classes, for example the class with the function to load from the different files the data. (DataManager, Settings).
5. Views: Classes that control all the outputs for screen, there are two ways to display information in the screen, through the console or with the graphic interface. (ApplicationView, ClienView, OrderView, ProductView).

We also have distributed the code in 4 parts:

- Marc: Application.
- Albert: Products.
- Bernat: Client.
- Aleix: Order and OrderLine.

Project Specifications

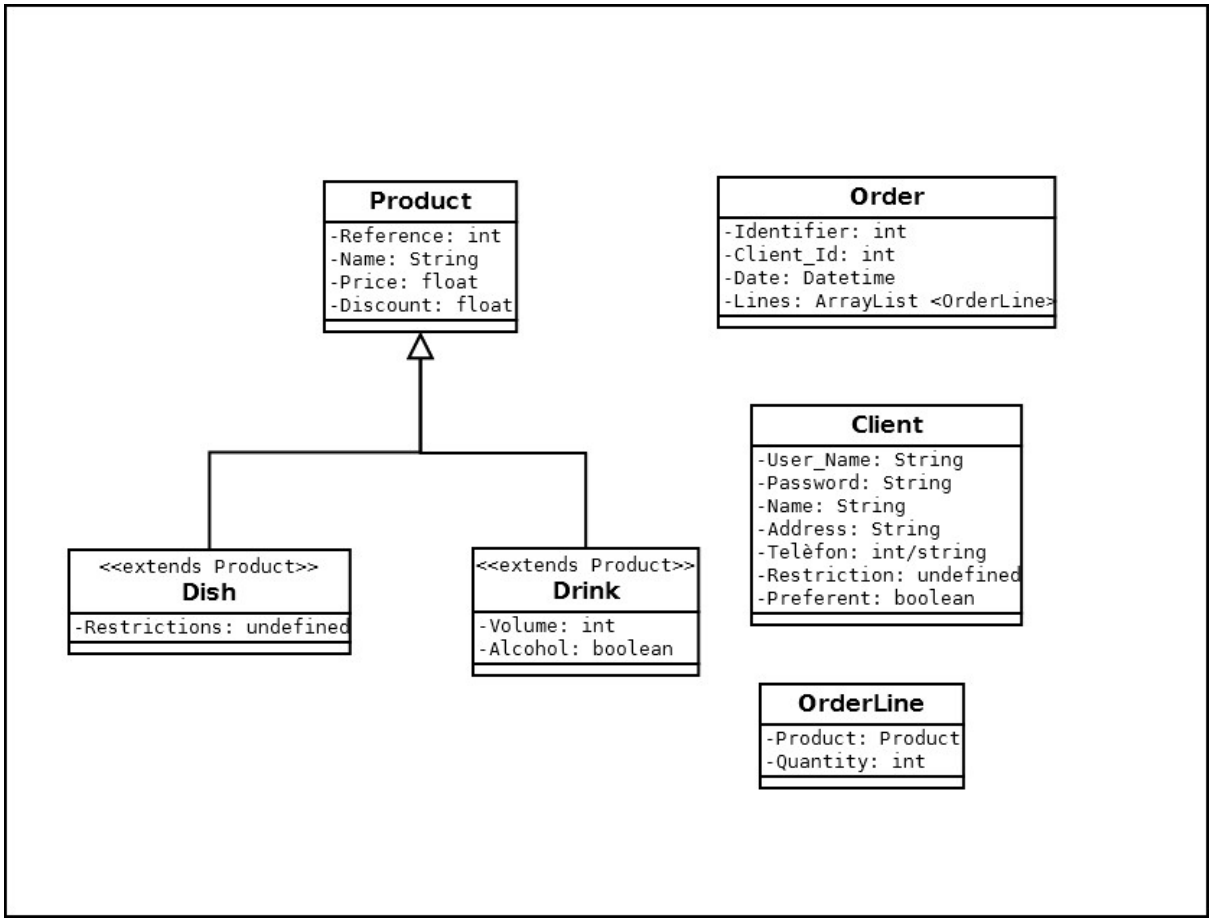
Java Application Requirements

- Generate at least 2 packages to order the code.
- Generate specific exceptions to manages the program exceptions.
- Manages Exceptions of the user actions

Considered functions to be accessible from the user side

- Add new products defined as type Dish or drink.
- Load Products from Data File.
- Delete product by ID.
- Display Information of a product by its ID.
- Create new client with the register data.
- Show all Commands for a client, ordered from newest to older.
- Create a new order for a client and adding the orderLines.
- Copy an existing order and generate a new one from the selected one.
- Load orders and clients from a different file for each one.
- Save orders and clients to a different file for each one.

Class attributes Specification from the statement of *Practica3*



Design

Package and Classes Design

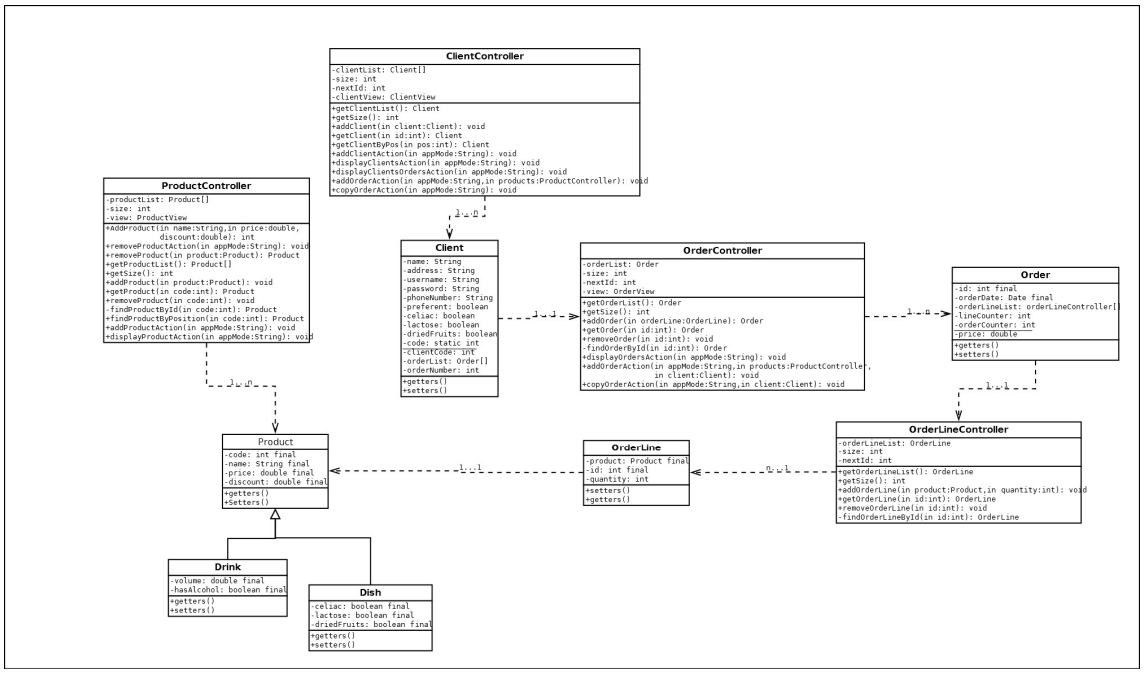
During the first meeting with the director, all of us agreed to develop this in a MVC (Model-View-Controller) scheme and using interfaces in our classes.

The application structure is the following (starting by the packages structure):

- **headers**: Includes only interfaces used to implement the classes.
    - **Controllers**: Includes controllers interfaces.
    - **Models**: Includes models interfaces.
    - **Utils**: Includes utils interfaces.
    - **Views**: Includes views interfaces.
- **grup06_practica3**: Includes the implementation for the classes using headers package interfaces:
    - **Controllers**: Includes the classes which will control the actions that we can execute in the model classes:
        - **ApplicationController (Mcabre)**: Called by the Main manages the application in general. Contains an instance of ApplicationView, an instance of ProductController and an instance of ClientCotroller.
        - **ClientCotroller (Bbosca)**: Contains a list of clients, all the actions needed to implement the functionality and an instance of the ClientView.
        - **OrderController (Amarine)**: Contains a list of orders, all the actions needed to implement the functionality and an instance of the OrderView.
        - **OrderLineController (Amarine):** Contains a list of order lines and all the actions needed to implement the functionality.
        - **ProductController (Mcabre)**: Contains a list of products, all the actions needed to implement the functionality and an instance of the ProductView.
    - **Events**: Contains all the classes for the implemented events that will be called in the GUI interfaces.
    - **Exceptions**: Contains all the classes for the team defined exceptions that will be thrown when the developers consider that is needed.
    - **Utils**: Contains generic files that we haven't considered to be part of the MVC:
        - **Settings (mcabre)**: Contains static variables with the aim to make the code more readable and more configurable.
        - **DataManager (mcabre)**: Implements all the functions related with the read/write from files. Data Files are located in the root of the project, but the path can be configured in the Settings.java file.
    - **Views**: Contains all the implementation for the views. Each file contains the text version and the JFrame version. The controller related to this class is the responsible to choose which version have to be launched.
        - **ApplicationView (mcabre)**: contains the menu to execute the actions required.

- **ClientView (Bbosca):** Contains all the user interface for client related actions and to grant the access to each client's orders.
- **OrderView (Amarine)**: Contains all the user interface for order and order line related actions.
- **ProductView (Acañellas)**: Contains all the user interface for product related actions.

Application UML Diagram



Development

Main (Mcabre)

Main file launches the application passing as a parameter the mode:

- **Settings.TEXT_MODE**: Console Mode.
- **Settings.GUI_MODE**: JFrame views

```
public class Main {

	public static void main(String[] args) {
		ApplicationController application = new ApplicationController();
		application.run(Settings.TEXT_MODE);
	}
}
```

Classes product, dish and drink (ACañellas)

The class product is an abstract class, that has in inheritance the classes dish and drink.
Class product have:
-The variables: int Code (the code of the product), String Name (the name of the product), double Price (the price of the product), double Discount (the discount that will be applicable to the price, in case to be a preferential client).
-Methods: the getters of the variables.

```
public abstract class Product implements Product_H{

    /**
     * Unique identifier of the product
     */
    protected final int code;

    /**
     * Name of the product
     */
    protected final String name;

    /**
     * Price of the product
     */
    protected double price;

    /**
     * Discount of the product if the client is vip
     */
    protected double discount;

    /**
     * Default constructor of the Product class
     *
     * @param code     unique identifier of the product
     * @param name     name of the product
     * @param price    price of the product
     * @param discount discount of the product if the client is vip
     */
    public Product(int code, String name, double price, double discount){
        this.code=code;
        this.name=name;
        this.price=price;
        this.discount=discount;
    }
```

Class dish have:
-The variables: the variables of the parent plus, Boolean Celiac (if the dish is not able for celiac), Boolean Lactose (if the dish is not able for lactose intolerants), Boolean Dried Fruits (if the dish is not able for people allergic to dried fruit).
-Methods: the getters of the variables.

```
public class Dish extends Product implements Dish_H {

    /**
     * Celiac?
     */
    protected final boolean celiac;

    /**
     * Lactose intolerance ?
     */
    protected final boolean lactose;

    /**
     * Dried Seeds allergy?
     */
    protected final boolean driedFruits;

    /**
     * Default constructor of the Plate class
     *
     * @param code     unique identifier of the product
     * @param name     name of the product
     * @param price    price of the product
     * @param discount discount of the product if the client is vip
     */
    public Dish(int code, String name, double price, double discount, boolean celiac, boolean lactose, boolean driedFruits){
        super(code, name, price, discount);
        this.celiac=celiac;
        this.lactose=lactose;
        this.driedFruits=driedFruits;
    }
```

Class drink have:
-The variables: double Volume (volume of the drink), Boolean hasAlcohol (if the drink has alcohol).
-Methods: the getters of the variables.

```
public class Drink extends Product implements Drink_H{

     /**
     * Volume of the Drink
     */
    private double volume;

    /**
     * Drink has alcohol?
     */
    private final boolean hasAlcohol;

    /**
     * Default constructor of the Drink class
     *
     * @param code       unique identifier of the product
     * @param name       name of the product
     * @param price      price of the product
     * @param discount   discount of the product if the client is vip
     * @param volume     volume of the Drink
     * @param hasAlcohol drink has alcohol?
     */
    public Drink(int code, String name, double price, double discount, double volume, boolean hasAlcohol){
        super(code, name, price, discount);
        this.volume = volume;
        this.hasAlcohol = hasAlcohol;
    }

    /**
     * Gets the volume of the drink
     *
     * @return volume of the drink
     */
    public double getVolume() {
        return volume;
    }
```

Class ProductView (ACañellas)

showProductsText();

Show the products by console, calls findProducByPosition this method returns a instance of the class product, and use its toString to print the information of the product.

```
/**
 * Method that shows the products list by console
 * @throws ProductNotFoundException
 */
public void showProductsText() throws ProductNotFoundException {
    for(int i=0; i<productController.getSize(); i++){
        System.out.println(productController.findProductByPosition(i));
    }
}
```

showProductsGui();

Show the products by window(graphic interface), calls findProducByPosition this method returns an instance of the class product, and use its toString to print the information of the product with a label.

```
/**
 * Method that shows a product list by windows
 * @throws ProductNotFoundException
 */
public void showProductsGui() throws ProductNotFoundException{

    int i;
    JFrame window = new JFrame();
    Container mineContainer = window.getContentPane();
    window.setTitle("Show Products List");
    window.setContentPane(mineContainer);
    window.setDefaultCloseOperation(EXIT_ON_CLOSE);
    window.setSize(1200,300);
    mineContainer.setLayout(new FlowLayout());
    for(i=0;i<this.productController.getSize();i++){
        mineContainer.add(new JLabel(this.productController.findProductByPosition(i).toString()));
    }
    window.setLocation(100,0);
    window.setVisible(true);
}
```

addProductText();

Shows a little menu with two options and the client has to choose between to add a dish or a drink, depends from the option which the client had choose, the application demands all the parameters in order to add a drink or a dish. The dialog is shown by the console.

```
public void addProductText() throws ProductFullListException {
    int op=0, code=0, ok=0;
    double price=0, discount=0, volume=0;
    boolean celiac=false, lactose=false, driedFruit=false, hasAlcohol=false;
    String name=".", a=".";
    Product_H d = null;
    System.out.println("\tPlease select an option.");
    System.out.println("\tPress 1 to add a dish or 2 to add a drink.");
    while(op!=1 && op!=2){
        System.out.println("Introduce 1 or 2");
        try{
            op=Integer.parseInt(keyboard.nextLine());
        }
        catch(NumberFormatException e){
            System.err.println("Error: "+e);
        }
    }
    while(ok!=1){
    switch(op){
    case 1:
        System.out.println("\tIntroduce the parametres in order to add a new dish.");
        //Code
        while(ok!=1){
            System.out.println("\n\tCode:");
            try{
                code=Integer.parseInt(keyboard.nextLine());
                ok=1;
            }
            catch(NumberFormatException e){
                System.err.println("Error: "+e);
                ok=0;
            }
        }
        //Name
        ok=0;
        while(ok!=1){
            System.out.println("\n\tName:");
            try{
                name=keyboard.nextLine();
                ok=1;
```

addProductGui();
Shows a little menu with two buttons and the client has to choose between to add a dish or a drink, depends from the button which the client had choose it calls different methods which create a dialog and ask all the parameters in order to add a drink or a dish. The dialog is shown by different windows (graphic interface).

```
/**
 * Method that adds a products (drink/dish) depends of the button pressed
 * @throws ProductNotFoundException
 */
public void addProductGui() {

    JFrame window = new JFrame();
    Container productContainer = window.getContentPane();
    window.setTitle("Add Product");
    window.setContentPane(productContainer);
    window.setDefaultCloseOperation(EXIT_ON_CLOSE);
    window.setSize(400,150);

    Button dish= new Button("Dish");
    productContainer.add(dish);
    dishAction accio=new dishAction(this);
    dish.addActionListener(accio);

    Button drink= new Button("Drink");
    productContainer.add(drink);
    drinkAction accio1=new drinkAction(this);
    drink.addActionListener(accio1);

    productContainer.setLayout(new FlowLayout());

    window.setVisible(true);
    window.setLocation(500,250);
}
```

buttonDish();

The method which is called from the addProductGui(); when the button of dish is pressed, creates the dialog asking the parameters in order to add a dish, it check if the values introduced are correct.

```
public void buttonDish(){
    int option = 0;
    String name = null;
    int code=0;
    double  price=0, discount=0;
    boolean celiac=false, lactose=false, driedFruits=false, correct1=false, correct=false;
    Dish dish = null;
    while(!correct){
        while(!correct1){
            try{
                code = Integer.parseInt(JOptionPane.showInputDialog("Introduce products code:"));
                correct1=true;
            }
            catch(NumberFormatException e){
                JOptionPane.showMessageDialog(null, "You have entered a character. Please, try entered a number", "ATENCION!", JOptionPane.W
            }
        }
        name = JOptionPane.showInputDialog("Introduce products name:");
        correct1=false;
        while(!correct1){
            try{
                price = Double.valueOf(JOptionPane.showInputDialog("Introduce products price:"));
                correct1=true;
            }
            catch(NumberFormatException e){
                JOptionPane.showMessageDialog(null, "You have entered a character. Please, try entered a number", "ATENCION!", JOptionPane.W
            }
        }
        correct1=false;
        while(!correct1){
            try{
                discount = Double.valueOf(JOptionPane.showInputDialog("Introduce products discount:"));
                correct1=true;
            }
            catch(NumberFormatException e){
                JOptionPane.showMessageDialog(null, "You have entered a character. Please, try entered a number", "ATENCION!", JOptionPane.W
            }
        }
        option = JOptionPane.showConfirmDialog(null, "Are you celiac?", "QUESTION", JOptionPane.YES_NO_OPTION);
        celiac = (option==JOptionPane.YES_OPTION);
        option = JOptionPane.showConfirmDialog(null, "Are you lactose intolerant?", "QUESTION", JOptionPane.YES_NO_OPTION);
```

buttonDrink();

The method which is called from the addProductGui(); when the button of drink is pressed, creates the dialog asking the parameters in order to add a drink, it check if the values introduced are correct.

```
public void buttonDrink(){
    int option = 0;
    String name = null;
    int code=0;
    double  price=0, discount=0, volume=0 ;
    boolean correct1=false, hasAlcohol=false, correct=false;
    Drink drink = null;
    while(!correct){
        while(!correct1){
            try{
                code = Integer.parseInt(JOptionPane.showInputDialog("Introduce products code:"));
                correct1=true;
            }
            catch(NumberFormatException e){
                JOptionPane.showMessageDialog(null, "You have entered a character. Please, try entered a number", "ATENCION!", JOptionPa
            }
        }
        name = JOptionPane.showInputDialog("Introduce products name:");
        correct1=false;
        while(!correct1){
            try{
                price = Double.valueOf(JOptionPane.showInputDialog("Introduce products price:"));
                correct1=true;
            }
            catch(NumberFormatException e){
                JOptionPane.showMessageDialog(null, "You have entered a character. Please, try entered a number", "ATENCION!", JOptionPa
            }
        }
        correct1=false;
        while(!correct1){
            try{
                discount = Double.valueOf(JOptionPane.showInputDialog("Introduce products discount:"));
                correct1=true;
            }
            catch(NumberFormatException e){
                JOptionPane.showMessageDialog(null, "You have entered a character. Please, try entered a number", "ATENCION!", JOptionPa
            }
        }
        correct1=false;
        while(!correct1){
```

removeProductText();

Ask to the client the code of the product which is wanted to remove by console, with the code it's called a method which will remove the product from the list.

```
/**
 * Method to remove a product by the code, it ask to the client the code of the product that he/she wants to remove by console
 * */
public void removeProductText() throws ProductNotFoundException {
    int code=0, ok=0;
    while(ok!=1){
        System.out.println("\n\tIntroduce the code of the product to remove.");
        try{
            code=Integer.parseInt(keyboard.nextLine());
            ok=1;
        }
        catch(NumberFormatException e){
            System.err.println("Error: "+e);
            ok=0;
        }
    }
    this.productController.removeProduct(code);
}
```

removeProductGui();

Ask to the client the code of the product which is wanted to remove by window, with the code its called a method which will remove the product from the list.

```
/**
 * Method to remove a product by the code, it ask to the client the code of the product that he/she wants to remove by windows
 * */
public void removeProductGui() {
    String title="Remove product";
    int code;


    productContainer=getContentPane();
    productContainer.setLayout(new GridLayout(3,0));
    setTitle(title);
    //Button exitProducts=new Button("EXIT");
    //exitProducts.setBounds(MAXIMIZED_HORIZ, MAXIMIZED_VERT, 40, 20);


    JLabel descr1= new JLabel("Write the code of the product:");
    productContainer.add(descr1);
    JLabel descr2=null;
    text= new JTextField();
    Button remove= new Button("Remove");
    productContainer.add(remove);
    removeAction accio=new removeAction(this);
    remove.addActionListener(accio);
    productContainer.add(text);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    setSize(400,200);
    setVisible(true);
}
```

showProductText();

Show the information of a product which code was asked before trough console. And prints its information in the console.

```
/**
 * Method to show a product by the code, it ask to the client the code of the product that he/she wants to show by console
 * */
public void showProductText() throws ProductNotFoundException {
    int code=0, ok=0;
    while(ok!=1){
        System.out.println("\n\tIntroduce the code of the product to remove.");
        try{
            code=Integer.parseInt(keyboard.nextLine());
            ok=1;
        }
        catch(NumberFormatException e){
            System.err.println("Error: "+e);
            ok=0;
        }
    System.out.println(this.productController.getProduct(code));
    }
}
```

showProductGui();

Show the information of a product which code was asked before through a window.
And prints its information in other window.

```
/**
 * Method to show a product by the code, it ask to the client the code of the product that he/she wants to show by window
 * */
public void showProductGui() {
    String title="Show Products";

    productContainer=getContentPane();
    productContainer.setLayout(new GridLayout(3,0));
    setTitle(title);
    //Button exitProducts=new Button("EXIT");
    //exitProducts.setBounds(MAXIMIZED_HORIZ, MAXIMIZED_VERT, 40, 20);

    JLabel descr1= new JLabel("Write the code of the product:");
    productContainer.add(descr1);

    text= new JTextField();
    Button busca= new Button("Busca");
    productContainer.add(busca);
    actionButton accio=new actionButton(this);
    busca.addActionListener(accio);
    productContainer.add(text);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    setSize(400,200);
    setVisible(true);
}
```

buttonPressed1 and 2();

When the button busca is pressed, it prints the information with a label.

```
public void buttonPressed1(){
    JLabel descr2=null;
    try{
        int code1=Integer.parseInt(text.getText());
        descr2= new JLabel(this.productController.getProduct(code1).toString());
        productContainer.add(descr2);
    }catch (ProductNotFoundException e){
        System.err.println("Error: "+e);
    }
}

public void buttonPressed2(){
    JLabel descr2=null;
    try{
        int code1=Integer.parseInt(text.getText());
        descr2= new JLabel(String.valueOf(code1));
        productContainer.add(descr2);
        this.productController.removeProduct(code1);
    }catch (ProductNotFoundException e){
        System.err.println("Error: "+e);
    }
}
```

Class DataManager (MCabre)

The class data manager is the responsible for the read and write the data from file. Examples of files with data.

Clients.txt:

```
#Id;name;address;username;password;phone_number;celiac;lactose;driedFruits
1;Marc;C/SN;marc;qwerty;+34678904567;false;false;false
2;Aleix;C/SN;aleix;qwerty;+34678904567;true;false;false
3;Bernat;C/SN;bernat;qwerty;+34678904567;true;true;false
4;Albert;C/SN;albert;qwerty;+34678904567;true;true;true
```

Products.txt:

```
#Type;Id;Name;Price;Discount;Celica;Lactose;DriedFruits
DISH;1;Amanida;10.0;10.0;true;true;false
DISH;2;Bistec;20.0;10.0;false;false;false
DRINK;3;Cocacola;2.5;5.0;250.0;false
DRINK;4;Vi;4.0;5.0;1000.0;true
```

Orders.txt:

```
#Id;ClientId;orderDate;Price;[orderLines]
1;1;1481128277769;10.0;[1:1-2:2]
```

Lines started with "#" means that the line is commented and will not be read.

It implements following functions:

public ProductController_H loadProductsFromFile(String filename)

This functions reads the products from file and returns a ProductController which contains the list of products.

```
/**
 * Imports products form file
 *
 * @param filename name of the file
 * @return list of products
 * @throws FileNotFoundException    thrown if the file does not exist
 * @throws ProductFullListException thrown if the list is full
 */
@Override
public ProductController_H loadProductsFromFile(String filename)
                throws FileNotFoundException, ProductFullListException {
        String tmpLine;
        ProductController_H tempList = new ProductController();
        try (Scanner input = new Scanner(new File(filename))) {
                while (input.hasNextLine()) {
                        tmpLine = input.nextLine();
                        // If the line is not a comment line
                        if (!tmpLine.substring(0, 1).equals("#")) {
                                String[] fields = tmpLine.split(";");
                                if (fields.length == 8 || fields.length == 7) {
                                        switch (fields[0]) {
                                                case "DISH": {
                                                        tempList.addProduct(new Dish(Integer.parseInt(fields[1]), fields[2],
                                                                        Double.parseDouble(fields[3]), Double.parseDouble(fields[
                                                                        fields[5].equals("true"), fields[6].equals("true"), field
                                                        break;
                                                }
                                                case "DRINK": {
                                                        tempList.addProduct(new Drink(Integer.parseInt(fields[1]), fields[2],
                                                                        Double.parseDouble(fields[3]), Double.parseDouble(fields[
                                                                        Double.parseDouble(fields[5]), fields[6].equals("true")))
                                                        break;
                                                }
                                                default: {
                                                        // Type not recognized
                                                        break;
                                                }
                                        }
                                }
                        }
                }
                input.close();
        }

        return tempList;
}
```

public ClientController_H loadClientFromFile(String filename)

This function reads clients list from the file and returns a ClientController containing this client list.

```
/**
 * Imports clients from file
 *
 * @param filename name of the file
 * @return list of clients
 * @throws FileNotFoundException   thrown if the file does not exist
 * @throws ClientFullListException thrown if the list is full
 */
@Override
public ClientController_H loadClientFromFile(String filename)
                throws FileNotFoundException, ClientFullListException {
        String tmpLine;
        ClientController_H tempList = new ClientController();
        try (Scanner input = new Scanner(new File(filename))) {
                while (input.hasNextLine()) {
                        tmpLine = input.nextLine();
                        // If the line is not a comment line
                        if (!tmpLine.substring(0, 1).equals("#")) {
                                String[] fields = tmpLine.split(";");
                                if (fields.length == 9) {
                                        tempList.addClient((new Client(Integer.parseInt(fields[0]), fields[1], fields[2], fields[
                                                        fields[4], fields[5], fields[6].equals("true"), fields[7].equals("true"),
                                                        fields[8].equals("true"))));
                                }
                        }
                }
                input.close();
        }

        return tempList;
}
```

public void loadOrdersFromFile(ClientController_H clients, String filename, ProductController_H products)

This function reads the orders from the file and assign this orders to the instance of client in the ClientController which have the same ID as the read one from file.
For the OrderLines we check that the selected item exists in the products list and the saves the instance of the product in the orderList field.

```
/**
 * Imports orders from file
 *
 * @param clients  list of clients
 * @param filename name of the file
 * @return list of clients with orders
 * @throws FileNotFoundException      thrown if the file does not exist
 * @throws OrderFullListException     thrown if the list is full
 * @throws OrderLineFullListException thrown if the list is full
 * @throws ProductNotFoundException
 * @throws NumberFormatException
 * @throws ClientNotFoundException
 */
@Override
public void loadOrdersFromFile(ClientController_H clients, String filename, ProductController_H products)
        throws FileNotFoundException, OrderFullListException, OrderLineFullListException, NumberFormatException,
        ProductNotFoundException, ClientNotFoundException {
    String tmpLine;
    try (Scanner input = new Scanner(new File(filename))) {
        while (input.hasNextLine()) {
            tmpLine = input.nextLine();
            // If the line is not a comment line
            if (!tmpLine.substring(0, 1).equals("#")) {
                String[] fields = tmpLine.split(";");
                if (fields.length == 5) {
                    int id = Integer.parseInt(fields[0]);
                    int clientId = Integer.parseInt(fields[1]);
                    Date data = new Date(Long.parseLong(fields[2]));
                    double price = Double.parseDouble(fields[3]);
                    String lines_temp = fields[4].substring(1, fields[4].length() - 1);
                    String[] lines = lines_temp.split("-");
                    OrderLineController_H order_lines = new OrderLineController();
                    for (int j = 0; j < lines.length; j++) {
                        String[] columns = lines[j].split(":");
                        order_lines.addOrderLine(products.getProduct(Integer.parseInt(columns[0])),
                                Integer.parseInt(columns[1]));
                    }
                    clients.getClient(clientId).getOrderList().addOrder(new Order(id, data, order_lines, pric
                    Order_H order = clients.getClient(clientId).getOrderList().addOrder(order_lines.getOrderL
                    for (int i = 1; i < order_lines.getSize(); i++) {
                        order.getOrderLineList().addOrderLine(order_lines.getOrderLineList()[i].getProduc
                    }
                }
            }
        }
        input.close();
    }
}
```

public void writeProductsToFile(ProductController_H products, String filename)

This function save the product list received as parameter to the file checking if it's a dish or it's a drink.

```
/**
 * Exports products to file
 *
 * @param products list of products
 * @param filename name of the file
 * @throws IOException thrown if the file cannot be opened
 */
@Override
public void writeProductsToFile(ProductController_H products, String filename) throws IOException {
        Product_H product;
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename))) {
                writer.write(Settings.PRODUCTS_HEADER);
                for (int i = 0; i < products.getSize(); i++) {
                        product = products.getProductList()[i];
                        if (product instanceof Dish) {
                                writer.write("DISH;" + product.getCode() + ";" + product.getName() + ";" + product.getPrice() + "
                                                + product.getDiscount() + ";" + ((Dish) product).isCeliac() + ";"
                                                + ((Dish) product).isLactose() + ";" + ((Dish) product).isDriedFruits() + "\n");
                        }
                        if (product instanceof Drink) {
                                writer.write("DRINK;" + product.getCode() + ";" + product.getName() + ";" + product.getPrice() +
                                                + product.getDiscount() + ";" + ((Drink) product).getvolume() + ";"
                                                + ((Drink) product).hasAlcohol() + "\n");
                        }
                }
                writer.close();
        }
}

/**
```

public void writeClientToFile(ClientController_H clients, String filename)

This functions writes de client list received in the parameters to the selected file:

```
/**
 * Exports clients to file
 *
 * @param clients  list of clients
 * @param filename name of the file
 * @throws IOException thrown if the file cannot be opened
 */
@Override
public void writeClientToFile(ClientController_H clients, String filename) throws IOException {
        Client_H client;
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename))) {
                writer.write(Settings.CLIENTS_HEADER);
                for (int i = 0; i < clients.getSize(); i++) {
                        client = clients.getClientList()[i];
                        writer.write(client.getId() + ";" + client.getName() + ";" + client.getAddress() + ";"
                                        + client.getUsername() + ";" + client.getPassword() + ";" + client.getPhoneNumber() + ";"
                                        + client.isCeliac() + ";" + client.isLactose() + ";" + client.isDriedFruits() + "\n");

                }
                writer.close();
        }
}
```

public void writeOrdersToFile(ClientController_H clients, String filename)

This function saves the orders in each client in the clientList received by parameter into a file. All the orders are added sequentially adding the client id to the line.

```
/**
 * Exports list of orders to file
 *
 * @param filename name of the file
 * @throws IOException                thrown if the file cannot be opened
 * @throws OrderNotFoundException     if the order not exists
 * @throws OrderLineNotFoundException if the order line not exists
 */
@Override
public void writeOrdersToFile(ClientController_H clients, String filename)
                throws IOException, OrderNotFoundException, OrderLineNotFoundException {
        Client_H client;
        Order_H order;
        String line_to_write = "";
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename))) {
                writer.write(Settings.ORDERS_HEADER);
                for (int i = 0; i < clients.getSize(); i++) {
                        client = clients.getClientList()[i];
                        for (int j = 0; j < client.getOrderList().getSize(); j++) {
                                order = client.getOrderList().getOrderList()[j];
                                line_to_write += order.getId() + ";" + client.getId() + ";" + order.getOrderDate().getTime() + ";
                                                + order.getPrice() + ";[";
                                for (int k = 0; k < order.getOrderLineList().getSize(); k++) {
                                        if (k > 0) {
                                                line_to_write += "-";
                                        }
                                        line_to_write += order.getOrderLineList().getOrderLineList()[k].getProduct().getCode() +
                                                        + order.getOrderLineList().getOrderLineList()[k].getQuantity();
                                }
                                line_to_write += "]";
                                writer.write(line_to_write + "\n");
                                line_to_write = "";
                        }
                }
                writer.close();
        }
}
```

ProductController (Mcabre)

The ProductController implements the list of products and manages the views related with the actions that the user is able to do with this products list.
The main functions are:

public void addProduct(Product_H product)
Implements the logic to add a product to the list or throws an exception if the list is full.

```
/**
 * Adds a new Product
 *
 * @param product product
 * @throws ProductFullListException thrown if list is full
 */
@Override
public void addProduct(Product_H product) throws ProductFullListException {
        if (size < productList.length) {
                productList[size] = product;
                size++;
        } else {
                throw new ProductFullListException();
        }
}
```

public void removeProduct(int code)
Implements the logic to remove a product from the list.

```
/**
 * Removes a product from the list
 *
 * @param code identifier of the product
 * @throws ProductNotFoundException thrown if the product does not exist
 */
@Override
public void removeProduct(int code) throws ProductNotFoundException {

        int i = 0;
        boolean found = false;

        // Find position of the product
        while ((i < size) && !found) {
                if (productList[i].getCode() == code) {
                        found = true;
                } else {
                        i++;
                }
        }

        if (found) {
                // Move products
                // for (int j = i; j < size - 1; j++) {
                //    productList[j] = productList[j + 1];
                // }
                // ArrayCopy(src, srcPos, dst, dstPos, length)
                if (i < size - 1) {
                        System.arraycopy(productList, i + 1, productList, i, size - 1 - i);
                } else {
                        productList[i] = null;
                }
                size--;
        } else {
                throw new ProductNotFoundException();
        }
}
```

Other functions:

- **getSize**: returns the size of the list.
- **findProductById**: Returns the product with this ID.
- **findProductByPosition**: returns the product in the selected position.
- **addProductAction**: call the view to add a product to the list.
- **removeProductAction**: call the view to remove a product from the list.
- d**isplayProductAction**: call the view to display the products.

ApplicationController (Mcabre)

The aim of the application controller is to manage the user petitions from the application view by calling the needed functions in the other controllers of the application. For this, I have implemented the following methods:

- **run(String ApplicationMode)**: runs the application deciding if will run as text or JFrame. This will be done in all controller when calling a view.
- **Actions from the view:**
    - addProductAction.
    - loadProductsFromFile.
    - removeProductAction.
    - displayProductAction.
    - loadClientsFromFile.
    - saveClientsToFile.
    - addClientAction.
    - displayClientOrderstAction.
    - displayClientsAction.
    - addOrderAction.
    - copyOrderAction.
    - throwExceptionAction: when the view wants to show an exception, calls this function.

ApplicationView (Mcabre)

The application View, basically implements the menus to access all the other actions that the user need to have access. We have two versions for all functions: The Text one and the GUI one:

runTextMenu()

Asks to the user for the option he wants to run.

```
/**
 * Display menu using the text mode
 */
@Override
public void runTextMenu() {
        try (Scanner scanner = new Scanner(System.in)) {
                int option;
                do {
                        System.out.println(" 1.  Add a new product");
                        System.out.println(" 2.  Load products from file");
                        System.out.println(" 3.  Remove product identified by code");
                        System.out.println(" 4.  Display product information");
                        System.out.println(" 5.  Create a new client");
                        System.out.println(" 6.  Display client orders");
                        System.out.println(" 7.  Add a new order");
                        System.out.println(" 8.  Copy an existing order");
                        System.out.println(" 9.  Load clients and orders from file");
                        System.out.println("10.  Save clients and orders into file");
                        System.out.println("11.  Exit");
                        System.out.println();
                        System.out.print("Option: ");

                        option = scanner.nextInt();
                        scanner.nextLine();

                        switch (option) {
                                case ADD_PRODUCT: {
                                        addProduct();
                                        break;
                                }
                                case LOAD_PRODUCTS: {
                                        loadProducts();
                                        break;
                                }
```

runGuiMenu()

Shows a button panel with all the actions available.

```
@Override
public void runGuiMenu() {
        iniComponents();
}

private void iniComponents() {
        this.setTitle("Restaurant Manager");

        // get our container
        Container container = getContentPane();

        // Creating a BorderLayout to manage our items
        container.setLayout(new GridLayout(5, 2));

        b_add_product.addActionListener(new AddProductButtonListener(this));
        b_load_products.addActionListener(new LoadProductsButtonListener(this));
        b_remove_product.addActionListener(new RemoveProductButtonListener(this));
        b_display_product.addActionListener(new DisplayProductButtonListener(this));
        b_add_client.addActionListener(new AddClientButtonListener(this));
        b_display_client_orders.addActionListener(new DisplayClientOrdersButtonListener(this));
        b_add_order.addActionListener(new AddOrderButtonListener(this));
        b_copy_order.addActionListener(new CopyOrderButtonListener(this));
        b_load_client_orders.addActionListener(new LoadClientOrdersButtonListener(this));
        b_save_client_orders.addActionListener(new SaveClientOrdersButtonListener(this));

        container.add(b_add_product);
        container.add(b_load_products);
        container.add(b_remove_product);
        container.add(b_display_product);
        container.add(b_add_client);
        container.add(b_display_client_orders);
        container.add(b_add_order);
        container.add(b_copy_order);
        container.add(b_load_client_orders);
        container.add(b_save_client_orders);

        // We need to free memory when we close the window
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        // We set the width and the height of our window.
        setSize(300, 300);
        // In other case, we won't be able to see the window.
        setVisible(true);
}
```

showExceptions(String message)

Shows the message received by parameter in the selected mode from the controller

```
public void showExceptionText(String text){
        System.out.println(text);
}

public void showExceptionGui(String text){
        JOptionPane.showMessageDialog(null, text, "Error", JOptionPane.ERROR_MESSAGE);
}
```

OrderListController (AMariné)

This class implements object OrderListController, which contain a list of OrderLine and the corresponding size. Also, this class include and identifier number (ID). OrderListController is a controller so have all the basic logical methods to control the list of OrderLine such as Add, remove and copy. This class has a toString method, getters and setters, too. They are detailed here:

```
public void addOrderLine(Product_H product, int quantity) throws
OrderLineFullListException
```

Adds a new OrderLine to OrderLineController. Needs a Product and a Quantity by parameter. If list is full, will throw OrderLineFullListException. It also increments size and nextId for the next OrderLine added.

```
/**
 * Adds a new OrderLine
 *
 * @param product  product
 * @param quantity quantity of the product
 * @throws OrderLineFullListException thrown if list is full
 */
@Override
public void addOrderLine(Product_H product, int quantity) throws OrderLineFullListException {
    if (size < orderLineList.length) {
        orderLineList[size] = new OrderLine(nextId, product, quantity);
        size++;
        nextId++;
    } else {
        throw new OrderLineFullListException();
    }
}
```

`public void removeOrderLine(int id) throws OrderLineNotFoundException`
Removes an OrderLine from the list. First, search for the OrderLine using its identifier number and then replaces it using System.arraycopy, filling the gaps at the same time. If there's only one element will fill it with null. Throws OrderLineNotFoundException if the orderline with the specified ID does not exist.

```
/**
 * Removes a order line from the list
 *
 * @param id identifier of the order line
 * @throws OrderLineNotFoundException thrown if the order line does not exist
 */
@Override
public void removeOrderLine(int id) throws OrderLineNotFoundException {

    int i = 0;
    boolean found = false;

    // Find position of the product
    while ((i < size) && !found) {
        if (orderLineList[i].getId() == id) {
            found = true;
        } else {
            i++;
        }
    }

    if (found) {
        if (i < size - 1) {
            System.arraycopy(orderLineList, i + 1, orderLineList, i, size - 1 - i);
        } else {
            orderLineList[i] = null;
        }
        size--;
    } else {
        throw new OrderLineNotFoundException();
    }
}
```

`private OrderLine_H findOrderLineById(int id) throws OrderLineNotFoundException`
Find an OrderLine using its ID. return

```
/**
 * Gets the orderLine identified by the id
 *
 * @param id identifier of the orderLine
 * @return orderLine identified by the id
 * @throws OrderLineNotFoundException thrown if the orderLine does not exist
 */
private OrderLine_H findOrderLineById(int id) throws OrderLineNotFoundException {
    for (int i = 0; i < size; i++) {
        if (orderLineList[i].getId() == id) {
            return orderLineList[i];
        }
    }
    throw new OrderLineNotFoundException();
}
```

This class also implements other setters and getters and the toString method.

Order (AMariné)

This class implements object Order. Order is the object that contains an OrderListController and its size, contains the price of the order and the date the order was created. Also, this class include an identifier number (ID).

Order is a model, so do not have any logical method, only getters and setters and a toString method.

OrderController (AMariné)

This class implements object OrderController, which contain a list of Order and the corresponding size. Also, this class include and identifier number (ID). OrderListController is a controller so have all the basic logical methods to control the list of OrderLine such as Add, remove and copy. This class has a toString method, getters and setters, too. They are detailed here:

`public Order_H addOrder(OrderLine_H orderLine) throws OrderFullListException, OrderLineFullListException`

`Adds a new order receiving an orderline. Throws OrderFullListException if the list is full. Also returns the order that you just added and increments size and the nextID.`

```
/**
 * Adds a new Order
 *
 * @param orderLine orderLine to add in the list
 * @return new Order
 * @throws OrderFullListException     thrown if list is full
 * @throws OrderLineFullListException thrown if list is full
 */
@Override
public Order_H addOrder(OrderLine_H orderLine) throws OrderFullListException {
    if (size < orderList.length) {
        orderList[size] = new Order(nextId);
        orderList[size].getOrderLineList().addOrderLine(orderLine.getProduct(), orderLine.getQuantity());
        size++;
        nextId++;

        return orderList[size - 1];
    } else {
        throw new OrderFullListException();
    }
}
```

public void removeOrder(**int** id) **throws** OrderNotFoundException
Removes an order using its identifier number. Throws OrderNotFoundException if there is no order with that identifier number. This method has same logical as removeOrderLine.

```
/**
 * Removes a order from the list
 *
 * @param id identifier of the order
 * @throws OrderNotFoundException thrown if the order does not exist
 */
@Override
public void removeOrder(int id) throws OrderNotFoundException {

    int i = 0;
    boolean found = false;

    // Find position of the product
    while ((i < size) && !found) {
        if (orderList[i].getId() == id) {
            found = true;
        } else {
            i++;
        }
    }

    if (found) {

        if (i < size - 1) {
            System.arraycopy(orderList, i + 1, orderList, i, size - 1 - i);
        } else {
            orderList[i] = null;
        }
        size--;
    } else {
        throw new OrderNotFoundException();
    }
}
```

public void addOrderAction(String appMode, ProductController_H products, Client_H client) **throws** OrderLineFullListException
New AddOrder (action)

```
/**
 * Adds a new orderAction depending on the application mode. Receive by parameter an instance of clients and the menu
 * @throws OrderLineFullListException if the list is full
 *
 */
public void addOrderAction(String appMode, ProductController_H products, Client_H client) throws OrderLineFullListException{
    if (appMode.equals(Settings.TEXT_MODE)) {
        view.createOrderText(client, products);
    } else {
        view.createOrderGui(client, products);
    }
}
```

public void copyOrderAction(String appMode, Client_H client) **throws** OrderNotFoundException

Copy a new order action

```
/**
 * Adds a new copyOrderAction depending o the application mode.
 * @throws OrderNotFoundException if there is no such order
 */
@Override
public void copyOrderAction(String appMode, Client_H client) throws OrderNotFoundException {
    if (appMode.equals(Settings.TEXT_MODE)) {
        view.copyOrderText(client);
    } else {
        view.copyOrderGui(client);
    }

}
```

public void displayOrdersAction(String appMode)

Creates a new action of displaying an order

```
/**
 * Display order depending on the application mode
 */
public void displayOrdersAction(String appMode){

    if (appMode.equals(Settings.TEXT_MODE)) {
        view.showOrdersText();
    } else {
        view.showOrdersGui();
    }
}
```

OrderView(AMariné)

This class implements the interface of the program (interaction with the user). Has six method, two per function: one for the graphical mode, and the other for the console mode. These two methods are detailed here:

`public void showOrderText(int ID) throws OrderNotFoundException`
Shows the order in a text mode using its ID. Throws OrderNotFoundException

```
/**
 * Display a text of the Order that corresponds with the identifier
 *
 * @throws OrderNotFoundException thrown if the client does not exist
 */
@Override
public void showOrderText(int ID) throws OrderNotFoundException {
    try {
        System.out.println(this.orderController.getOrder(ID));
    } catch (OrderNotFoundException e) {
        System.err.println("Error: " + e);
    }

}
```

`public void showOrderGui(int ID) throws OrderNotFoundException`
Shows the order in a text mode using its ID. Throws OrderNotFoundException

```
/**
 * Display an order that corresponds with the identifier in a graphical mode
 *
 * @param ID id of the order we want to print
 * @throws OrderNotFoundException thrown if the order does not exist
 */
@Override
public void showOrderGui(int ID) throws OrderNotFoundException {
    JFrame window = new JFrame();
    Container Container1 = window.getContentPane();
    window.setTitle("Display Order");
    window.setContentPane(Container1);
    window.setDefaultCloseOperation(EXIT_ON_CLOSE);
    window.setSize(1200, 200);
    window.setLocation(100, 0);
    window.setVisible(true);
    Container1.setLayout(new FlowLayout());
    try {
        Container1.add(new JLabel(this.orderController.getOrder(ID).toString() + "\nIdentifier: " +
    this.orderController.getOrder(ID).getId() + " \nDate: " + this.orderController.getOrder(ID).getOrderDate()
    + " \nPrice: " + this.orderController.getOrder(ID).getPrice() + " \nNumber of products: " +
    this.orderController.getOrder(ID).getOrderLineList().getSize()));
    } catch (OrderNotFoundException e) {
        System.err.println("Error: " + e);
    }
}
```

`public void showOrdersText()`
Show all the orders in a text mode

```
/**
 * Display a text of the List of Order from the history of a client
 */
@Override
public void showOrdersText() {
    for (int i = orderController.getSize()-1; i >= 0; i--) {
        System.out.println(orderController.getOrderList()[i]);
    }

}
```

public void showOrdersGui()

Show all the orders in a graphical mode

```
/**
 * Display a text of the List of Order from the history of a client
 *
 * @param client instance of client
 */
@Override
public void showOrdersGui() {
    JFrame window = new JFrame();
    Container Container1 = window.getContentPane();
    window.setTitle("Display Orders");
    window.setContentPane(Container1);
    window.setDefaultCloseOperation(EXIT_ON_CLOSE);
    window.setSize(1200, 200);
    window.setLocation(100, 0);
    window.setVisible(true);
    Container1.setLayout(new FlowLayout());
    for (int i = orderController.getSize()-1; i >= 0; i--) {
        Container1.add(new JLabel(orderController.getOrderList()[i].getOrderLineList().getOrderLineList().toString()
                + "\nIdentifier: " + orderController.getOrderList()[i].getId() + " \nDate: " + orderController.getOrderList()[i]
                        .getOrderDate() +" \nPrice: " + orderController.getOrderList()[i].getPrice() + " \nNumber of products: "
                + orderController.getOrderList()[i].getOrderLineList().getSize()));
    }
}
```

public void createOrderText(Client_H client, ProductController_H menu) **throws** OrderLineFullListException

The code of this method is in the annex.
This method creates a new order interacting with the user in a text mode using a bucle that has the next steps:

- First shows a menu and displays it.
- Asks the user which product he wants to add.
- Asks the user which quantity he wants to add.
- An orderline is created and then added to a new order.
- Prove that the product that just has been added does not have any allergen which the client is allergic to. If this situation happens, notifies the user and gives the possibility to remove this product.
- Asks the user if wants to add another product in his order, if he says yes, repeat the previous steps, if not, goes to next line.
- Displays the order that client is demanding by the moment and asks if he wants to confirm.
- If the order is confirmed shows the ID of the order and exit, if not restart the buckle.

public void createOrderGui(Client_H client, ProductController_H menu) **throws** OrderLineFullListException

The code of this method is in the annex.
This method creates a new order interacting with the user in a graphical mode using a bucle that has the next steps:

- First shows a menu and displays it.
- Asks the user which product he wants to add.
- Asks the user which quantity he wants to add.
- An orderline is created and then added to a new order.
- Prove that the product that just has been added does not have any allergen which the client is allergic to. If this situation happens, notifies the user and gives the possibility to remove this product.
- Asks the user if wants to add another product in his order, if he says yes, repeat the previous steps, if not, goes to next line.

- Displays the order that client is demanding by the moment and asks if he wants to confirm.
- If the order is confirmed shows the ID of the order and exit, if not restart the buckle.

```
public void copyOrderText(Client_H client) throws OrderNotFoundException
Code of this method is in the annex.
Copy an existing order into a new one.
```

- Displays a list of orders that the client made.
- Asks the user which order he wants to copy.
- Asks for confirmation and displays the confirmation page.

```
public void copyOrderGui(Client_H client) throws OrderNotFoundException
Code of this method is in the annex.
Copy an existing order into a new one.
```

- Displays a list of orders that the client made.
- Asks the user which order he wants to copy.
- Asks for confirmation and displays the confirmation page.

ClientController (BBosca)

The ClientController implements the list of clients and manages the views related with the actions that the user is able to do with this products list.
The main functions are:

public void addClient(Product_H client)
Implments the logic to add a client to the list or throws an exception if the list is full.

```
/**
 * Adds a new Client
 *
 * @param client client
 * @throws ClientFullListException thrown if list is full
 */
public void addClient(Client_H client) throws ClientFullListException {
    if (size < clientList.length) {
        clientList[size] = new Client(nextId, client.getName(), client.getAddress(), client.getPhoneNumber(),
                client.getUsername(), client.getPassword(), client.isCeliac(), client.isLactose(), client.isDriedFruits());
        size++;
        nextId++;
    } else {
        throw new ClientFullListException();
    }

}
```

public void displayClientsOrdersAction(String appMode)
Show the orders that corresponds with the id that the user selected.

```
/**
 * Display the orders that corresponds with the id that the user want to know the orders of the client
 *
 * @param appMode
 * @throws ClientNotFoundException
 */
@Override
public void displayClientsOrdersAction(String appMode) throws ClientNotFoundException {
    int id = -1;
    if (appMode.equals(Settings.TEXT_MODE)) {
        id = clientView.selectIdClientListText();

    } else {
        id = clientView.selectIdClientListGui();
    }
    if(id >=0){
        this.getClient(id).getOrderList().displayOrdersAction(appMode);
    }
}
```

Other functions:

- **getClientList**: returns the list of the clients.
- **getSize**: returns the size of the list.
- **getClient**: returns the client with this id.
- **getClientByPos**: returns the client in the selected position.
- **addClientAction**: call the view to add a client to the list.
- **addOrderAction**: add a new order on a client that is chosen by id.
- **displayClientsAction**: call the view to display the clients.
- **copyOrderAction**: copy a order on a client that is chosen by id.

ClientView (BBosca)

The ClientView implements the view of client ans list of clients by text and by graphical interface.
The main functions are:

public void displayClientListGui()
Display the client list by graphical interface.

```
/**
 * Display a text of the list of clients with graphical interface
 */
@Override
public void displayClientListGui() {
    int i;
    String space=" ";
    JFrame window = new JFrame();
    Container mineContainer = window.getContentPane();
    window.setTitle("Display Client List");
    window.setContentPane(mineContainer);
    window.setSize(1200,300);
    mineContainer.setLayout(new FlowLayout());
    for(i=0;i<this.clientController.getSize();i++){
        mineContainer.add(new JLabel("Client:"+space+space+"identifier="+space+this.clientController.getCl
            +space+space+",address="+space+this.clientController.getClientList()[i].getAddress()+space+spa
            +space+space+",username="+space+this.clientController.getClientList()[i].getUsername()+space+s
            +space+space+",celiac="+space+this.clientController.getClientList()[i].isCeliac()+space+space+
            +space+space+",driedFruits="+space+this.clientController.getClientList()[i].isDriedFruits()+sp
    }
    window.setLocation(100,0);
    window.setVisible(true);
}
```

public void addClientText()
Created and add a new client on the client list by text.

```
/**
 * Create a new client and add client to list of clients (clientController)
 */
@Override
public void addClientText() {
    long id = 0;
    String name = null, address = null, phoneNumber = null, username = null, password = null, aux = null;
    boolean celiac=false, lactose=false, driedFruits=false, correct=false;
    Client consumer = null;

    while(!correct){
        System.out.println("\nIntroduce your name: ");
        name = keyb.nextLine();
        System.out.println("Introduce your address: ");
        address = keyb.nextLine();
        while(!correct){
            try{
                System.out.println("Introduce your phone number: ");
                id = keyb.nextLong();
                phoneNumber = String.valueOf(id);
                keyb.nextLine();        //Clean buffer
                correct=true;
            }
            catch(InputMismatchException e){
                keyb.nextLine();        //Clean buffer
                System.out.println("You have entered a character. Please, try entered a number.");
            }
        }
        System.out.println("Introduce your username: ");
        username = keyb.nextLine();
        System.out.println("Introduce your password: ");
        password = keyb.nextLine();
        System.out.println("Are you celiac?");
        aux = keyb.nextLine();
        celiac = aux.equals("yes");
```

public void addClientGui()

Created and add a new client on the client list by graphical interface.

```
/**
 * Create a new client and add client to list of clients (clientController)
 */
@Override
public void addClientText() {
    long id = 0;
    String name = null, address = null, phoneNumber = null, username = null, password = null, aux = null;
    boolean celiac=false, lactose=false, driedFruits=false, correct=false;
    Client consumer = null;

    while(!correct){
        System.out.println("\nIntroduce your name: ");
        name = keyb.nextLine();
        System.out.println("Introduce your address: ");
        address = keyb.nextLine();
        while(!correct){
            try{
                System.out.println("Introduce your phone number: ");
                id = keyb.nextLong();
                phoneNumber = String.valueOf(id);
                keyb.nextLine();        //Clean buffer
                correct=true;
            }
            catch(InputMismatchException e){
                keyb.nextLine();        //Clean buffer
                System.out.println("You have entered a character. Please, try entered a number.");
            }
        }
        System.out.println("Introduce your username: ");
        username = keyb.nextLine();
        System.out.println("Introduce your password: ");
        password = keyb.nextLine();
        System.out.println("Are you celiac?");
        aux = keyb.nextLine();
        celiac = aux.equals("yes");
```

Other functions:

- **displayClientText**: display the client that corresponds whit the id by text.
- **displayClientGui**: display the client that corresponds whit the id by graphical interface.
- **displayClientListText**: display the client list by text.
- **selectIdClientListText**: the user select an id of the one client of the client list that is shown by text.
- **selectIdClientListGui**: the user select an id of the one client of the client list that is shown by graphical interface.
- **closeKeyboardClientView**: close the keyboard of client view class.

Client (BBosca)

The Client class is the class that contains all the information and atributes of one client.

This are the methods of the class:

- **getId**: return the id of the client.
- **getName**: return the name of the client.
- **getAddress**: return the address of the client.
- **getPhoneNumber**: return the phone number of the client.
- **getUsername**: return the username of the client.
- **getPassword**: return the password of the client.
- **isLactose**: return if the client is lactose intolerant or no.
- **isDriedFruits**: return if the client is dried fruits allergic or no.
- **isCeliac**: return if the client is celiac or no.
- **isPreferential**: return if the client is a preferential client or no.
- **setPreferential**: set if the client is a preferential client.
- **getOrderList**: return order list of the client.
- **toString**: method toString of client class

Evaluations

Developer Evaluation

Each developer has been responsible of the testing of their assigned modules including local tests (which are not committed to GIT). Local tests include:

- Isolated Tests which doesn't depend on anything out of their components.
- Test sets to reproduce the user actions and possible user's errors.

Tests

Class	Method	Description	OK?
ClientController(BBosca)	getClientList	Gets the list of clients	OK
	getSize	Get the size of the client list	OK
	addClient	Add a new Client on the client list	OK
	getClient	Gets the client identified by the id	OK
	getClientByPos	Gets the client situated in the position introduced	OK
	addClientAction	Call the view of client to add a client to the list of clients	OK
	displayClientAction		
Product(ACañellas)	getCode	Gets the code	OK
	getName	Gets the name	OK
	getPrice	Gets the price	OK
	getDiscount	Gets the discount	OK
Dish(ACañellas)	isCeliac	Gets if the client is celiac	OK
	isLactose	Gets if the client is lactose intolerant	OK
	isDriedFruits	Gets if the client is allergic to dried fruits	OK
Drink(ACañellas)	getVolume	Gets the volume	OK
	gethasAlcohol	Gets if the drink has alcohol	OK
ProductView(ACañellas)	showProductsText	Show the list of products in the console	OK
	showProductGui	Show the list of products in the graphic interface	Ok
	showProductText	Display the data of a product in the console	Ok
	showProductGui	Display the data of a product in the graphic interface	Ok

	addProductText	Ask to the client the data in order to add a product in the console	Ok
	addProductGui	Ask to the client the data in order to add a product in the graphic interface	Ok
	removeProductText	Remove the product by code in the console	Ok
	removeProductGui	Remove the product by code in the graphic interface	Ok
DataManager(Mcabre)	loadProductsFromFile	Read From File when it exists, when not and with multiple errors	Ok
	loadClientFromFile	Read From File when it exists, when not and with multiple errors	Ok
	loadOrdersFromFile	Read From File when it exists, when not and with multiple errors	Ok
	writeProductsToFile	Save Data To File	Ok
	writeClientToFile	Save Data To File	Ok
	writeOrdersToFile	Save Data To File	Ok
OrderController(Amarine)	AddOrder	Adds a new Order	Ok
	RemoveOrder	Removes an Order	Ok
OrderLineController	AddOrderLine	Adds a new OrderLine	Ok
	RemoveOrderLine	Removes an OrderLine	Ok
OrderView	showOrderText	Shows an order in a text mode using its ID	Ok
	showOrderGui	Shows an order in a graphical mode using its ID	Ok
	showOrdersText	Shows a list of orders from a client in a text mode using its ID	Ok
	showOrdersGui	Shows a list of orders from a client in a graphical mode using its ID	Ok
	createOrderText	Creates a new Order in a text mode	Ok
	createOrderGui	Creates a new Order in a graphical mode	Ok
	copyOrderText	Copy a new Order in a text mode	Ok

	copyOrderGui	Copy a new Order in a graphical mode	Ok

Global Evaluation

Each developer have been responsible for the testing of their own modules after merging all the components to the master's branch in order to ensure that during the merge we solved all the conflicts without affecting the modules.
Developers have been asked to test other developers modules in the global application in order to check the robustness of the solution and check possible errors that haven't been considered by the module's developer.

Conclusions

Thanks to this project we have seen how the professional world works, the different problems that you will have to deal with. For example when the difference programmers join their parts and conflicts appears, and you will need to solve them working with your partners in order to finish correctly the project.

We have seen also, how to structure a program to make it more scalable and understandable by all the parts which take part on it.

Problems:
Also we had some problems with understanding how to implement the graphic interface, defining the properties to the different windows or the containers.

We had to face to misunderstandings between us and correct them while developing the software without waiting for a new meeting and trying to not delay our parts in order to avoid delaying other developers.

We have respected the deadlines imposed by our leaders, and we also have attended more or less to all the meetings.

Resources used and annexes

Development environment

Eclipse IDE for Java Developers (not an specific version).

Software Versioning and revision control

GIT with Github.com platform with following branches:

- Master: Code merged from all development branches.
- Examples: MVC with interfaces example from RCasanovas to use as a starting point.
- Prog1: Mcabre development branch.
- Prog2: Acañellas development branch.
- Prog3: Bbosca development branch.
- Prog4: Amarine development branch.

Each developer decided to use its own decision GIT manager from this ones:

- GitHub application.
- Tortoise GIT.
- SmartGit.

Java Documentation

Java Javadocs for general information about functions included in each classes.