UNIT 1. FINAL EXERCISE

CateringFX

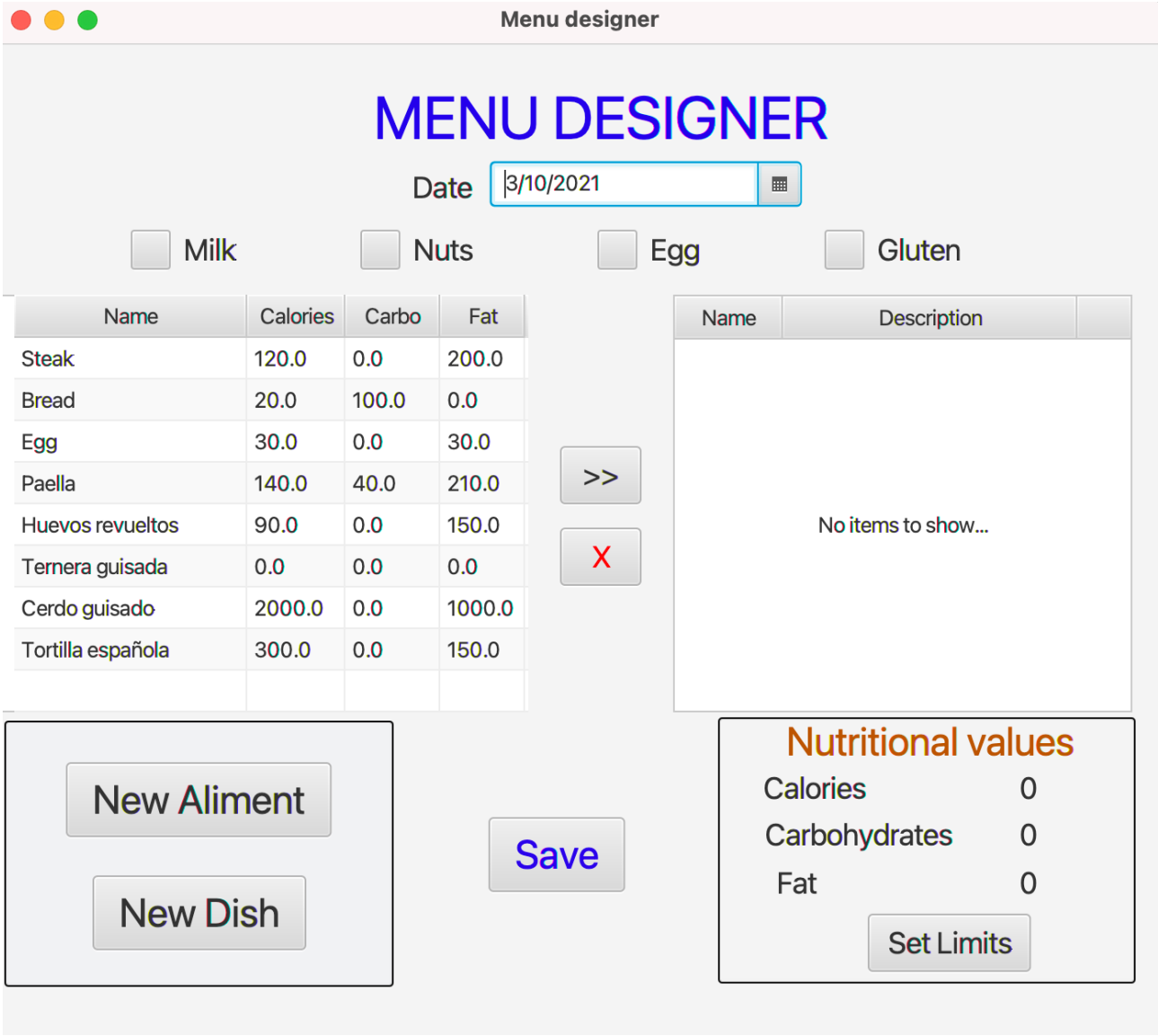
Service and process programming

MARIA CONSUELO RUBIO
IES SAN VICENTE

INDEX

1. Introduction and first steps

In the final exercise of this unit, we are going to implement a JavaFX application to manage the creation of menus in a company of catering. The appearance of the application will be something like this:



1.1Setting up the project

We are going to create a JavaFX project called **CateringFX**:

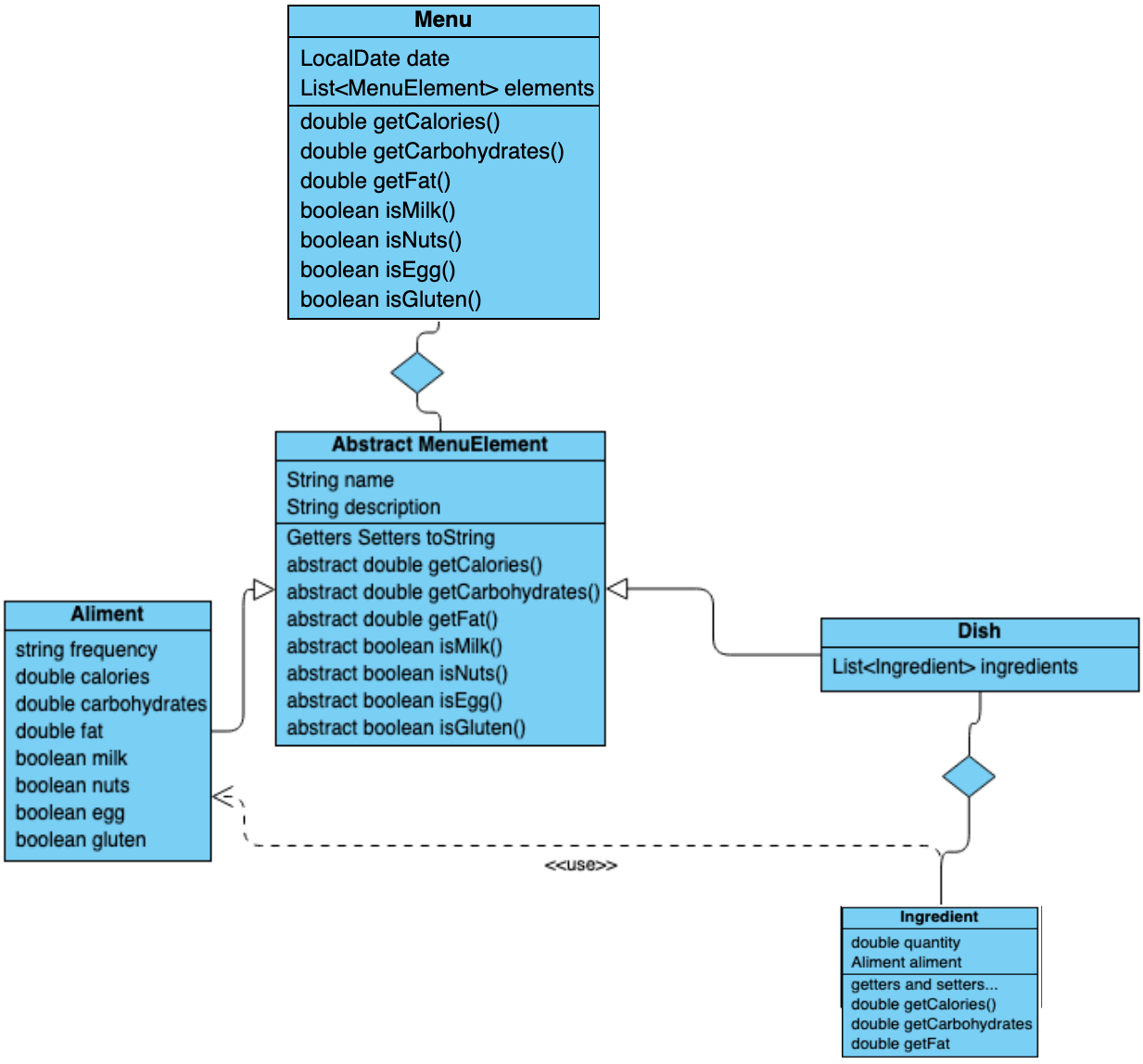
Once the project is created, inside the *Source Packages* section, create the following packages and subpackages:

- *cateringfx* package, which will be our main package with the JavaFX main class and controllers.
- *cateringfx.model* package, to store our models (the classes that will be explained later)
- *cateringfx.utils* package, to store some useful classes.

2. Class structure

Besides the JavaFX main application with the FXML file and controller (we will see them in next section), we are going to need some additional classes to store the information about the menus, dishes, and aliments.

2.1 The classes in model package:



As you can see in the diagram above, you have a class corresponding to the menu of a date which have a list of elements (they can be aliments such as bread or a dish such as paella). Both kind of elements (dish and aliment) inherit from an abstract class where there are defined some abstract methods that are needed to implement in their children.

A Dish has a list of ingredients which are aliments and the quantity used in this recipe. So to implement the abstract methods in Aliment it will be simple getter but in dish, you will have to calculate the amount of calories, carbohydrates and fat of all the aliments in each quantity.

2.2 FileUtils class

In order to get and save the information from / to text files, we are going to create a class called **FileUtils** in the cateringfx.*utils* package. This class will have a set of useful static methods, such as:

- `public static List<MenuElement> loadElements()` → to load the aliments and dishes stored in aliments.txt and dishes.txt.

- `public static void storeAliment(Aliment a)` → to store a new Aliment defined in the application.
- `public static void storeDish(Dish d)` → to store a new Dish defined in the application.

- `public static void storeMenu(Menu m)` → to store a menu in the file called *dateOfTheMenu*.menu

.2.1.1. Files of the project

The structure of the text file containing the aliments(aliments.txt) will be:

Name:description;frequency;gluten;milk;nuts;egg;calories;carbohydrates;fat

For instance:

```
Steak;Pork steak;Weekly;false;false;false;false;120.0;0.0;200.0
Bread;Wheat bread;Daily;true;false;false;false;20.0;100.0;0.0
Egg;Eggs;Weekly;false;false;false;true;30.0;0.0;30.0
```

The structure of the text file containing the dishes(dishes.txt) will be:

nameOfTheDish;descriptionOfTheDish;Ingrediente1Quantity;Ingredient1Name;Ingredient1Description; Ingrediente1Frequency; Ingrediente1Gluten; Ingrediente1Milk; Ingrediente1Nuts; Ingrediente1Egg; Ingrediente1Calories; Ingrediente1Carbohydrates; Ingrediente1Fat; Ingrediente2Quantity;Ingredient2Name;Ingredient2Description; Ingrediente2Frequency; Ingrediente2Gluten; Ingrediente2Milk; Ingrediente2Nuts; Ingrediente2Egg; Ingrediente2Calories; Ingrediente2Carbohydrates; Ingrediente2Fat…

…

For example:

```
Paella;Paella tradicional;2.0;Pollo;Pollo de
campo;Dayly;true;false;false;false;10.0;20.0;5.0;4.0;Arroz;Paella
2;Dayly;false;false;false;true;30.0;0.0;50.0
Huevos revueltos;Huevos revueltos con aceite de oliva;3.0;Huevo;Prueba
2;Dayly;false;false;false;true;30.0;0.0;50.0
Ternera guisada;Falda de ternera tierna;
Cerdo guisado;Carne de cerdo tierna;20.0;Carne de cerdo;Carne de cerdo
tierna;Weekly;false;false;false;false;100.0;0.0;50.0
Tortilla española;huevos;6.0;huevos;hue-
vos;Daily;false;false;false;true;50.0;0.0;25.0
```

And the structure of the text file containing the menu(dateOfTheMenu.menu) will be:

Date;aliments of dishes in the same format that before.

Recommendation: implement toString of Aliment, Dish and Menu to follow this format so in the function to write the files you only will need to call println(object).

.2.2. Other useful classes

Although it is not compulsory, it may be useful to add some other classes. For instance, a class called *MessageUtils* (in the *cateringfx.utils* package) to show different *Alert* messages. It could have these static methods:

- static void showError(String header, String message) to show error messages.
- static void showMessage(String header, String message) to show information messages.
- ...

You can add as many classes as you need inside this package.

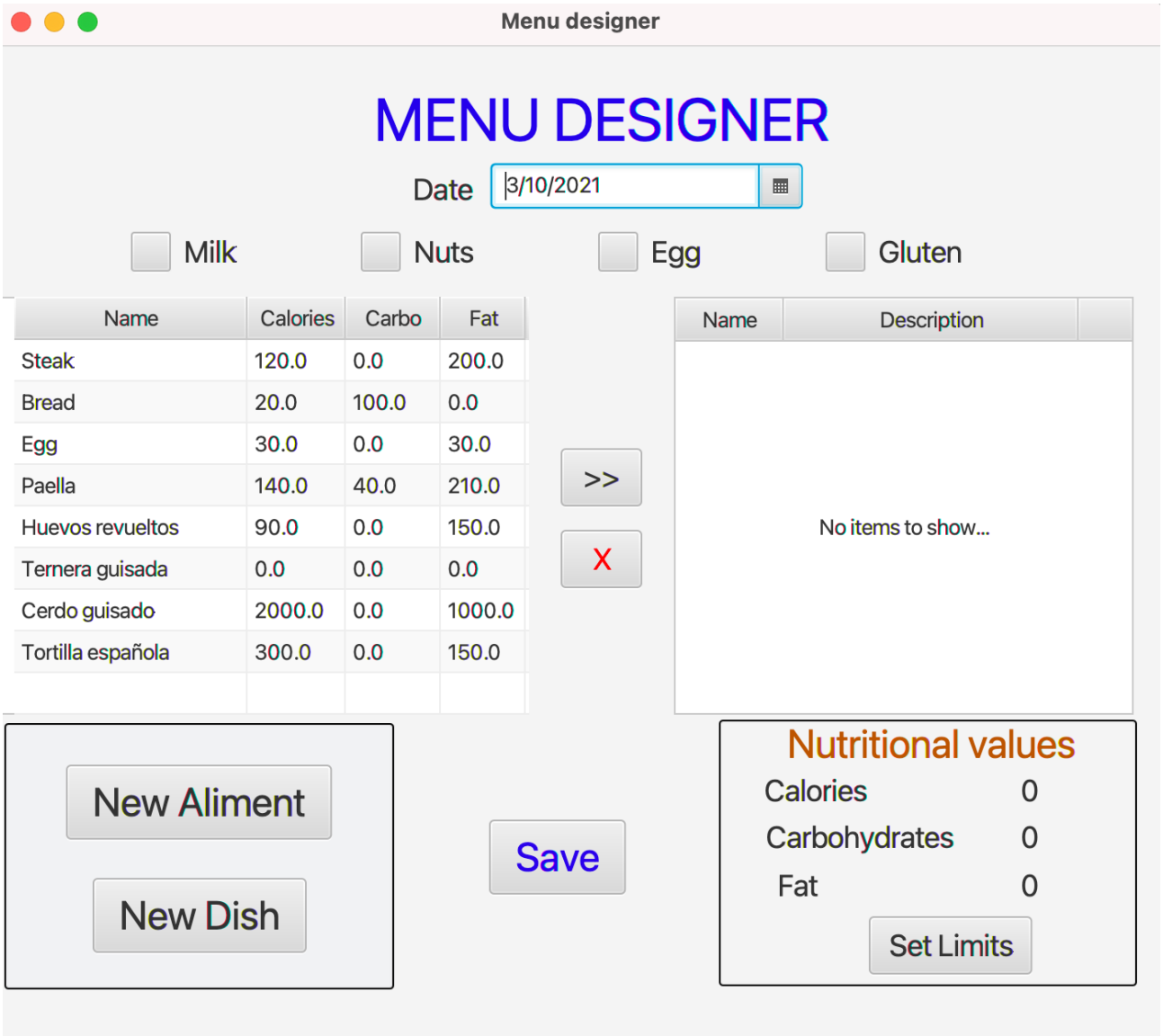
3. The JavaFX application

Let's create the JavaFX application classes. Follow these steps:

1. The JavaFX Main Application will be called **MainApplication** inside *summercampfx* package.
2. There will be an FXML file called **main-view.fxml** with its associated controller (**MainController.java**). As the project is created using Maven, the fxml files will be in the resources directory and the java files will be in the cateringfx package (or com.cateringfx package).
3. Make sure that your MainApplication class loads the contents from the FXML file.

.3.1. Designing the main view

Use now Scene Builder to design the main scene and get an appearance similar to this:



You can, for instance, use one gridPane to arrange general distribution, and in each cell of the grid a Vbox, Hbox or SplitPane.

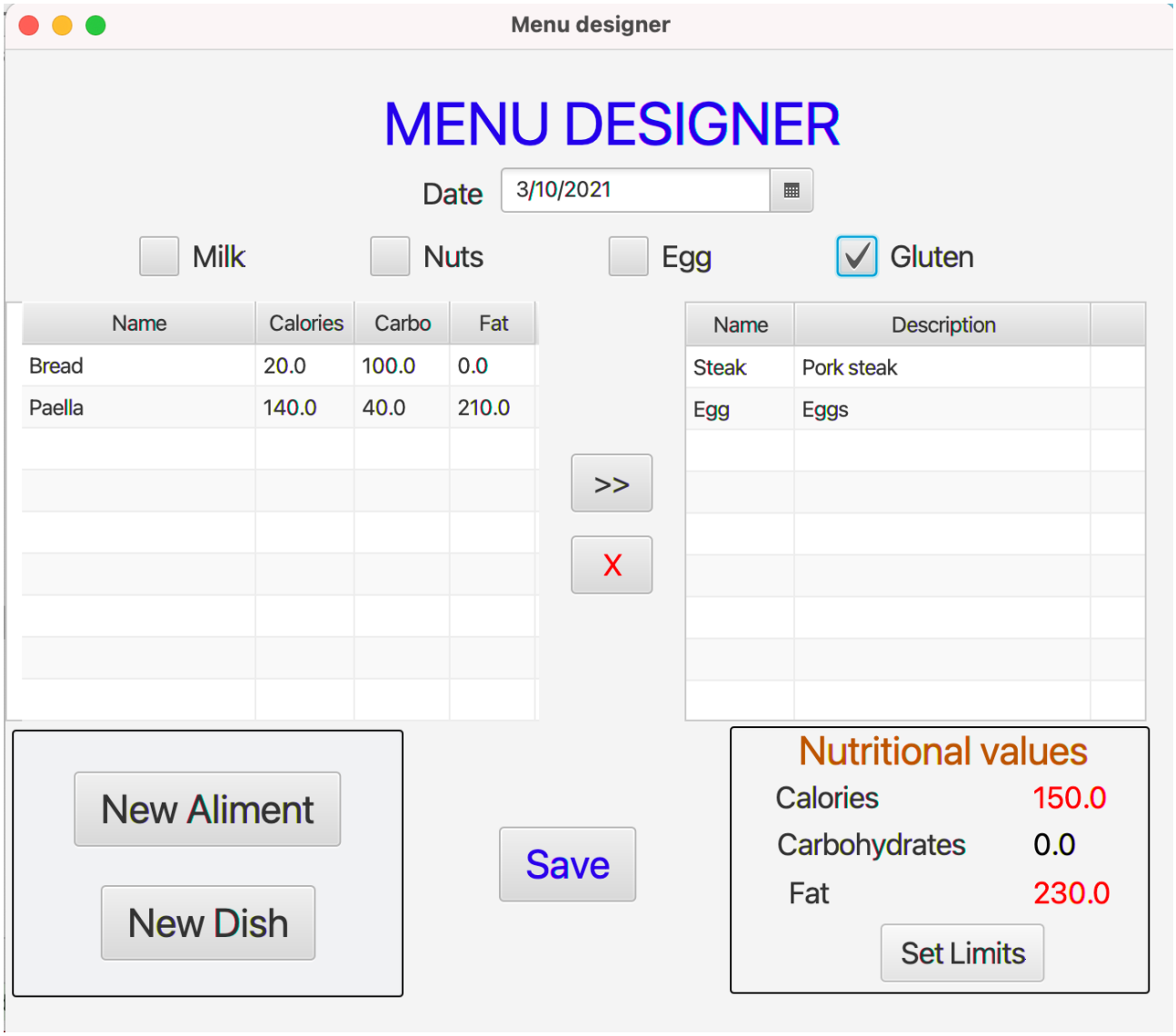
Remember to set an *fx:id* to each element that may need to be accessed from the controller.

3.2 How should it work?

As soon as the application starts, it will need to load the files aliments.txt and dishes.txt and place them into the left table of the view. After that the user will be able to pick a date for the menu and select some of the elements on the left to move to the right table that will be the elements that finally will be in the menu. Every time we add a new element to the menu (the table of the right) the nutritional values of below must be updated.

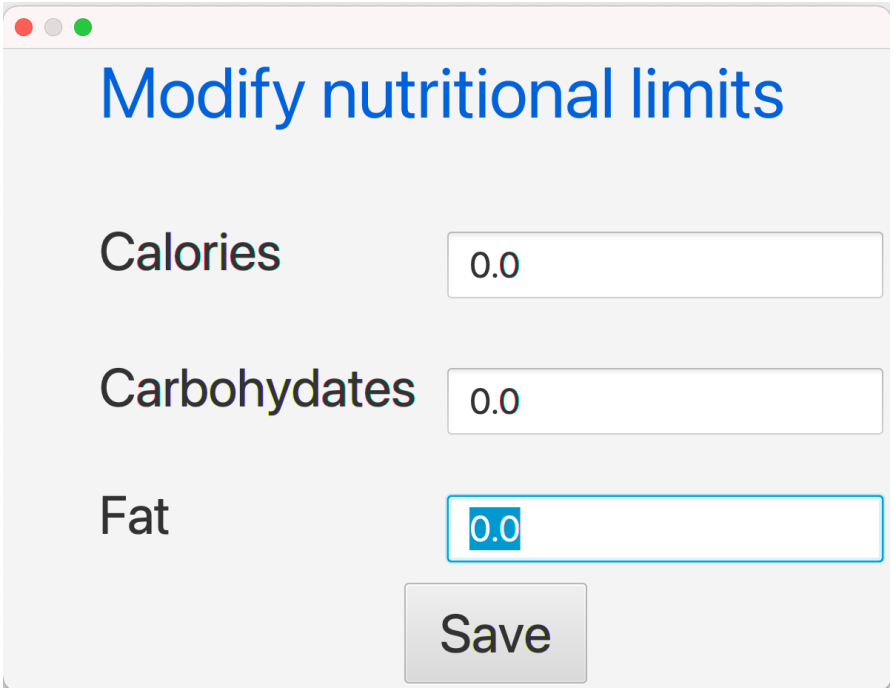
With the checkboxes the user will be able to choose element who have or not these allergens. Only one or none of the checkboxes can be selected at a time. The result of the consult of these allergens will be made with **streams** in order to show only the selected elements in the left table.

It can be possible too to remove elements of the right column.



3.1.1 Setting the nutritional values limits

We can set the limits of the nutritional values with the button “Set Limits”. When we set these limits, the values that exceed them will appear in red. These limits will be set in a new stage and when we click on the button save the window will be closed and we return to the main view where the values of the limits will be updated. (You can import static variables between java files to achieve this easily).

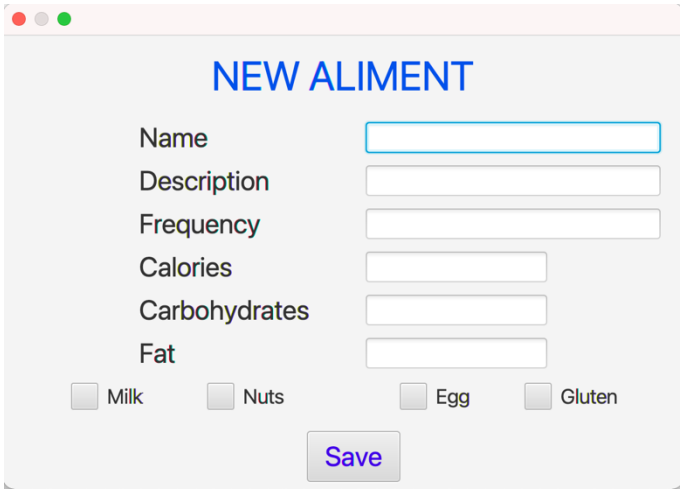


3.1.2 Creating new Aliments

We can create new aliments using the corresponding button “New Aliment” in the main view. This button will open a new stage where we will be able to add new aliments and when we click on the *save* button, we will add this aliment to the *aliment.txt* file. (At the end of the file). Only if all the text fields have been completed will you able to add this new aliment to the file, if there is any field without information you will write an error message. On the other hand, if everything is ok, you will write a message of success.

After saving an ailment, the information will be erased, and we will be able to add more new aliments until we close the window.

When we exit from this stage, we will update the left table of the main view with the new aliments created. You must read again the files to fill the left table.

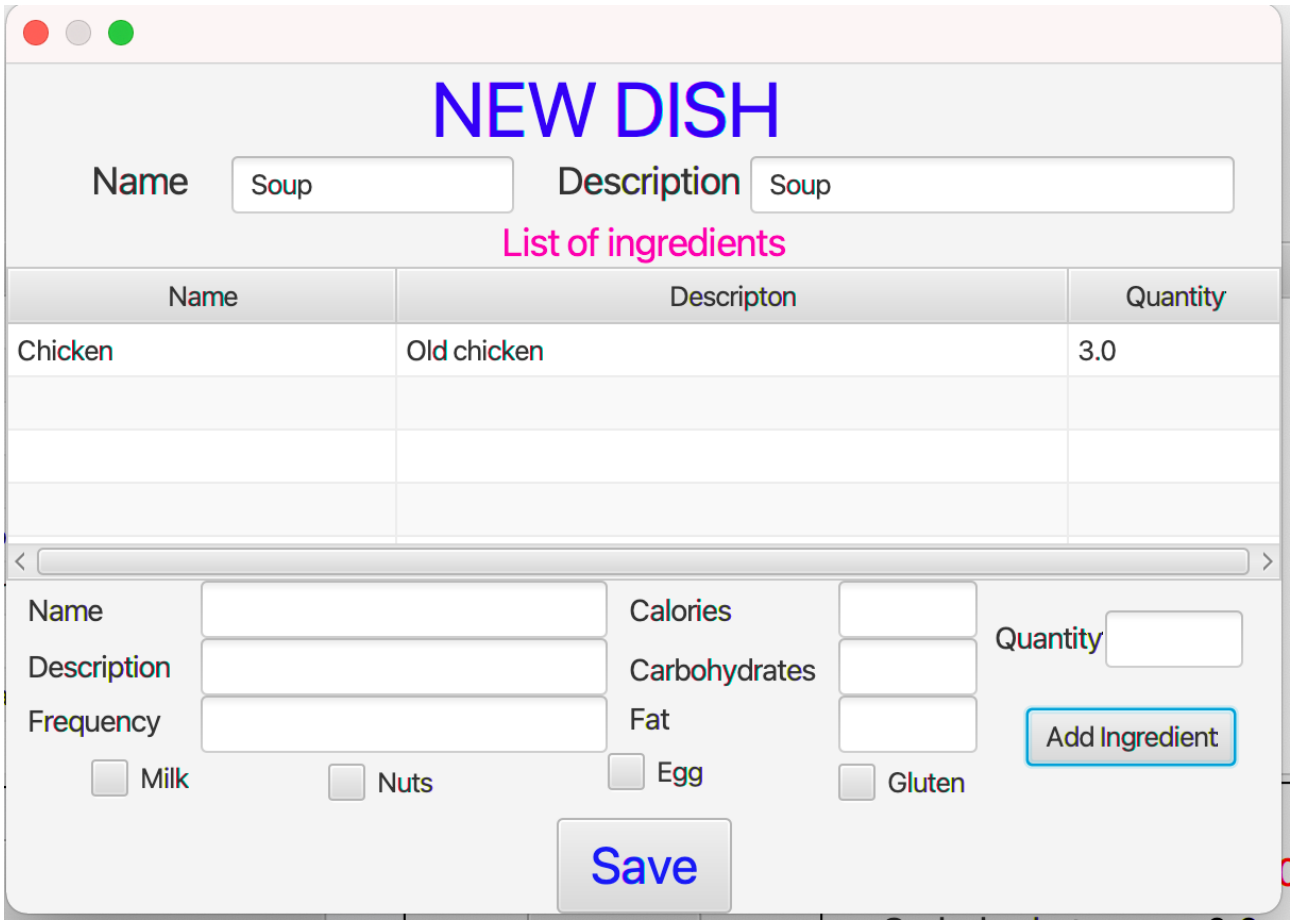


3.1.3 Creating new Dishes (optional)

In this new stage we will be able to create new dishes. Once we have completed the name and the description of the dish, we will be able to add new Ingredients to the list. If we don't have the name or the description of the dish, we will show an error message.

We will be able to add ingredients to the dish (only if all the text fields have information) with the *Add Ingredient* button and after that we will save the dish in the file *dishes.txt* when we click on the *save* button.

When we return to the main view, we will update the list of elements (the left table) to have the new dishes created.



3.1.4 Saving the menu

After having every element of our menu in the table of the right, we can save the menu clicking on the *save* button. We will save a new file called with the date of the menu and the extension .menu.

4. Optional improvements

Besides the compulsory part of this exercise, you can try to implement this optional part.

4.1 Adding CSS styles (1 point)

Add a CSS stylesheet with some default styles for the main view, such as:

- Background color: choose among any color other than gray, black or white.
- Font color: according to the background chosen, it can be either white or gray.
- Every button must have a black background with white text color.
- Other styles that you may want to add.

4.2 Another optional part (1 point)

The stage to create a **new dish** is not compulsory, it will count as an optional part.

5. Evaluation rules

5.1 Compulsory part

To get your final mark, the following rules will be applied, and you have **to use streams and lambda expressions wherever you can. (The more the merrier)**

Item achieved	Points
Class structure (*model* and *utils* packages)	1
JavaFX application layout, similar to the one shown in previous figures. You must use the appropriate layout containers to arrange the controls properly in the view (HBox and/or VBox are very helpful).	1,5
Loading existing dishes and aliments from the text files and writing in them using streams.	1
Showing and updating nutritional values, and the stage to setting them	0,5
Adding and removing elements to or from the table of the right	1
Choose the elements regarding the checkboxes using streams	1,5
Add new aliments in a different stage.	1,5
Showing error messages whenever something can't be done.	1
Code documentation (Javadoc comments for every class and public method or constructor), cleanliness and efficiency.	1

5.2 About the optional improvements

If you decide to implement any optional improvement for this exercise, keep in mind that:

- If your compulsory part is lower than 10, you can get up to 10 points by implementing one (or many) of the optional improvements (up to 1 point per improvement)
- If your compulsory part is perfect (10 points), you can get up to 11 points by implementing ALL the optional improvements.

5.3 Where you can use streams

Here you have the methods and occasions I used streams in my project, try to use them in the same places if you can:

- In the methods **getCalories**, **getCarbohydrates**, **GetFat**, **isMilk**, **isGluten**, **isEgg**, **isNuts** in the classes **Dish** and **Menu**.
- In the method **loadElements** in the class **FileUtils**
- In the methods corresponding to the event of **clicking on the checkboxes** of Milk, Egg, Nuts and Gluten. To filter in the table of the left only the elements which have the characteristics needed.