UNIVERSIDAD
PANAMERICANA

Patrones de diseño

Materia: Patrones de diseño

Profesor: Lidia Chávez López

Carrera: Ingeniería en Inteligencia Artificial

Alumno: David Gamaliel Arcos Bravo

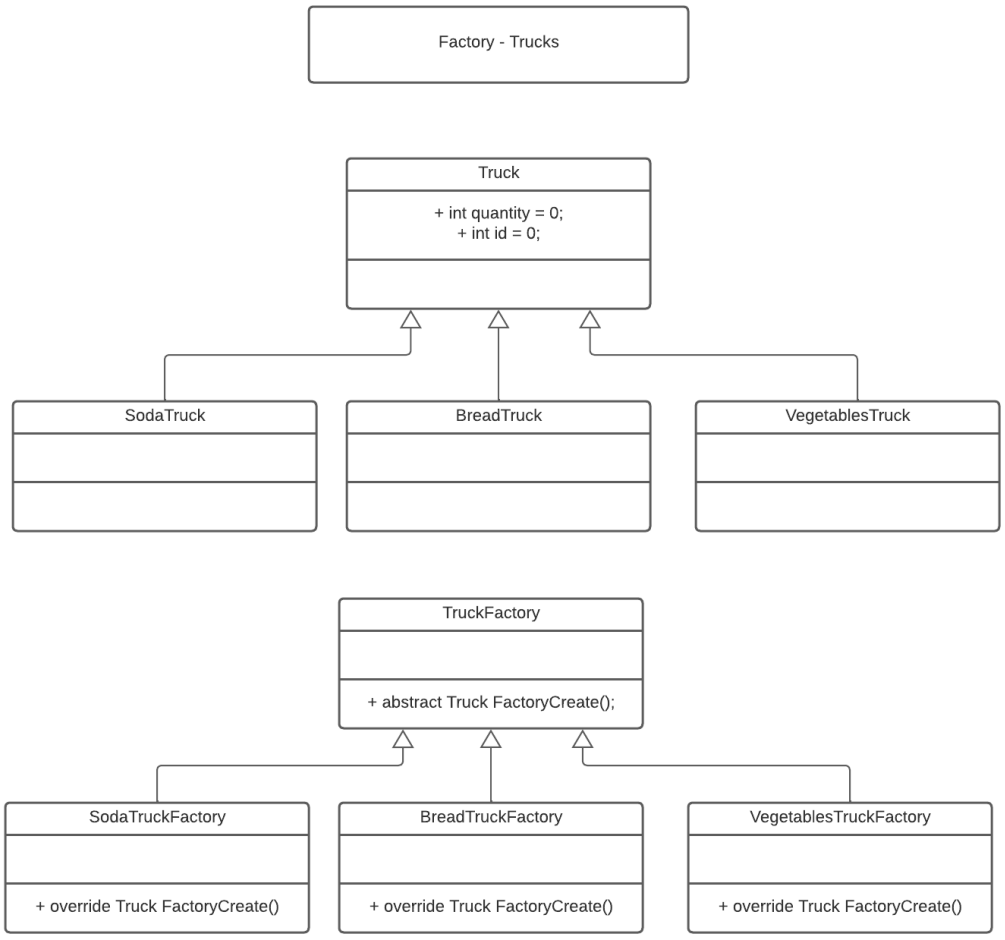
Fecha: 11 / 05 / 2022

In this project, four distinct design patterns were used, including two creational, one behavioral and one structural.

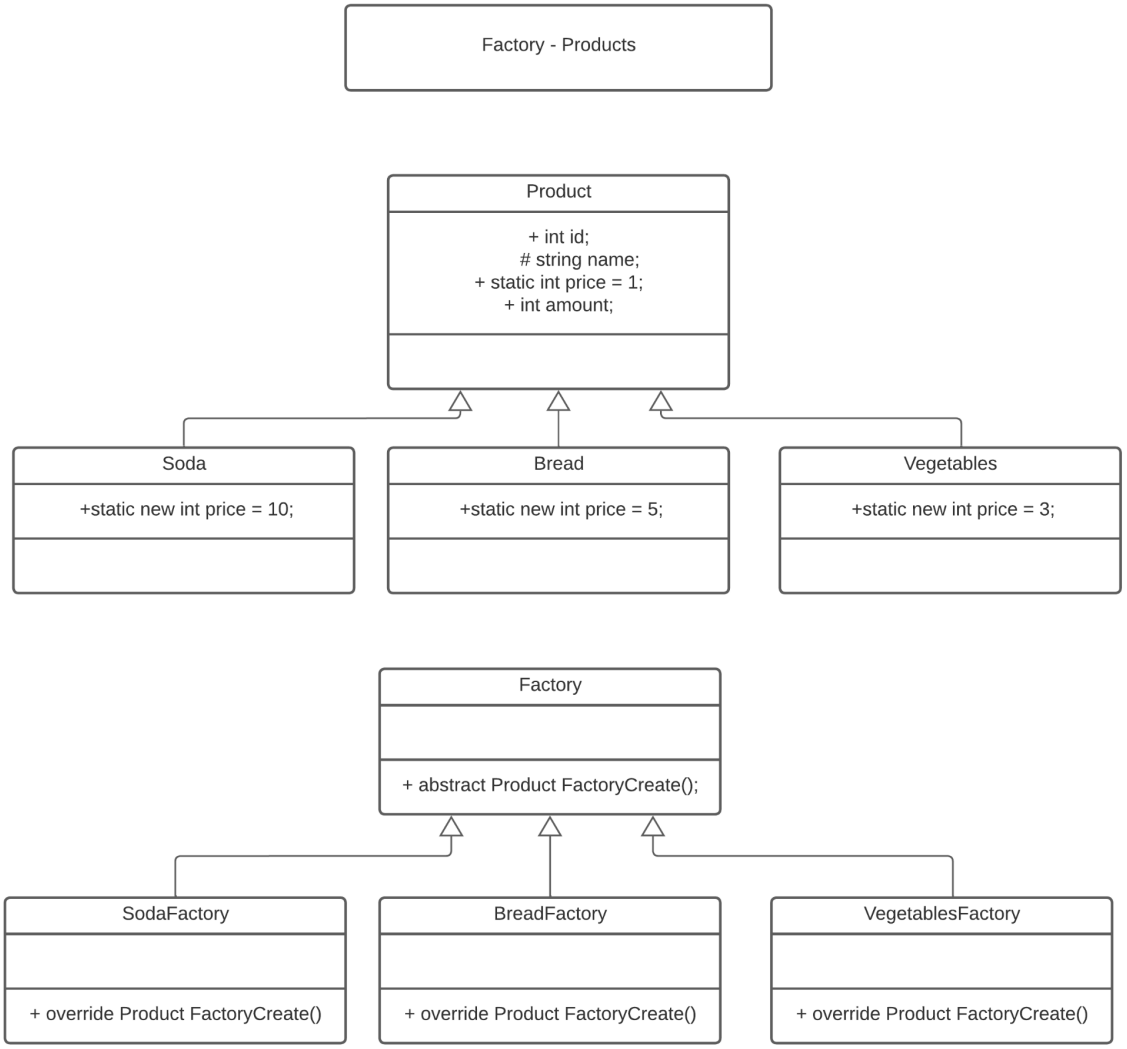
Selected patterns were:
Creational -> builder, **factory**, abstract factory, **singleton**
Behavioral -> bridge, decorator, **facade**, proxy
Structural -> **mediator**, observer, interpreter, command

Factory

Factory pattern has two implementations in this project, one regarding the creation of the products and the other regarding the creation of the different distribution trucks.



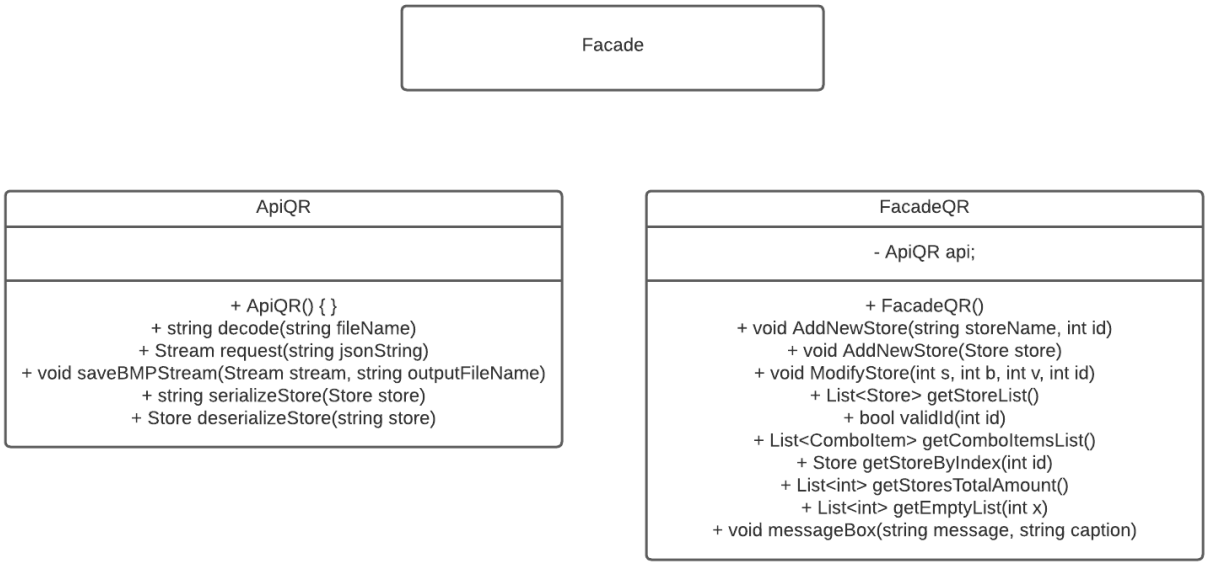
Factory pattern was used in this case to create the different types of trucks, where each truck has their specific type of product to deliver, as well as a defined quantity of products of capacity for each truck. Trucks were created with these specifications into the corresponding factories.



Factory pattern was used in this case to create the different types of products that we can order. In this case each factory was in charge of creating the three different kinds of products, assigning to each one their corresponding values regarding price and name of the product.

Facade

Due to the constant access to the qr lecture during the process of our app, it was convenient to create a higher level structure to access the information stored in the qr images. That’s where the facade pattern goes on.



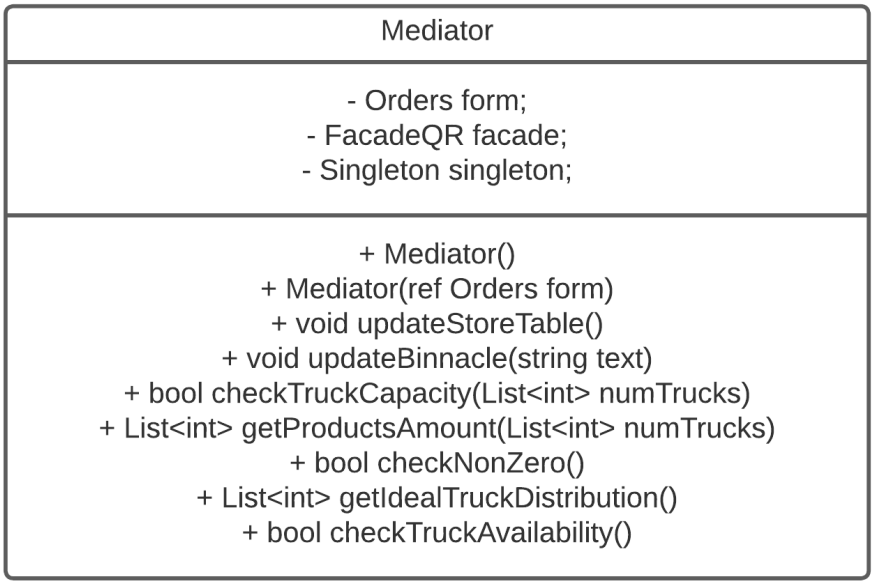
In our application, the class ApiQR is in charge of the operations using the API to decode and encode the store information into the qr and get it into classes again, as well as the process of serialization and parsing the json data provided by the api. Nevertheless, some operations using these functions can get very complex as we need to keep frequently asking for the data information stored into the qr images, so that's where the class FacadeQR comes into the game.

This class FacadeQR provides us several useful pre-built functions to access the data stored by just calling the functions and getting it formatted in a useful manner. This way we don’t have to access the api or the qr’s themselves but to call a function well all the complexity keeps reduced to receiving a list or a useful values and doing some stuff with it.

Between the most useful functions cames getStoreByIndex, function that returns the store corresponding to some id, AddNewStore and ModifyStore, to create and change store qr images respectively to modify the information about the stores, or getStoreList, the function that return the full information of all the qr codes into a list of stores ready to use. All of these functions without needing to access directly to the api or api qr functions, just by getting those implementations.

Mediator

Mediator gives us a middle point class to interact between other classes, important at the moment of managing multiple forms and classes, as well as the GUI.



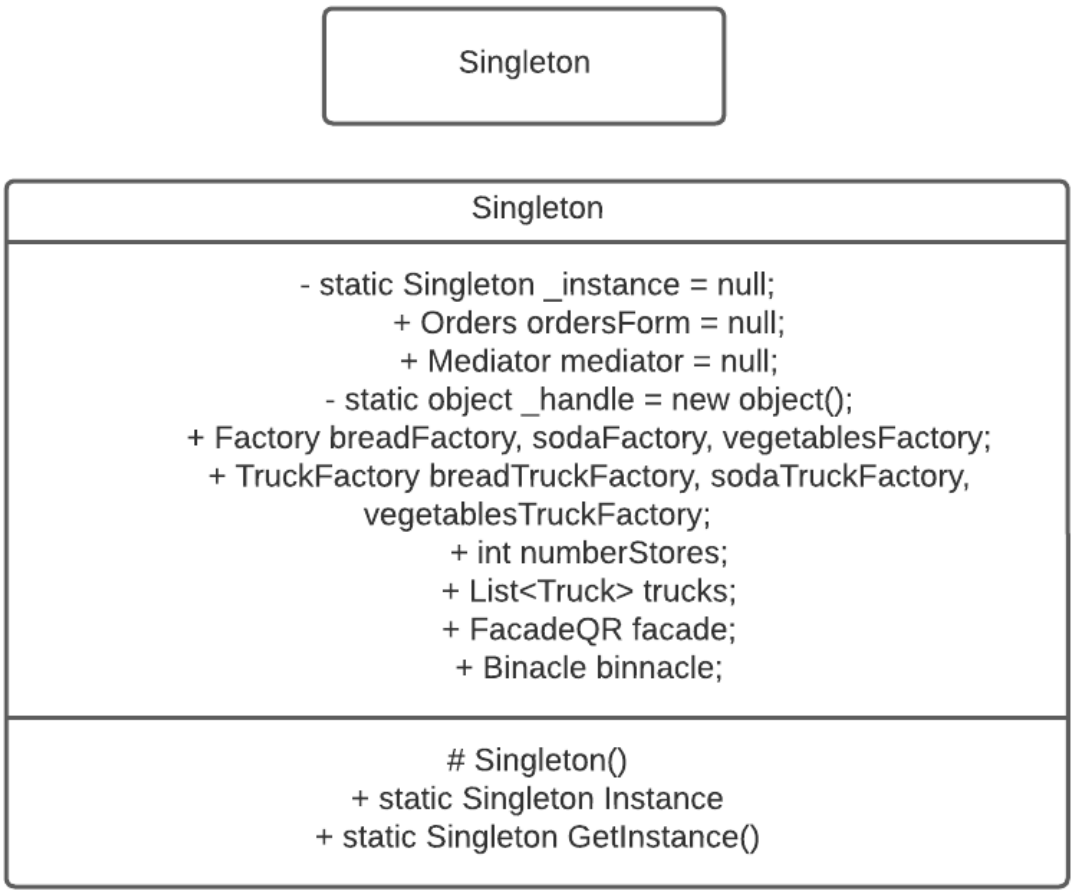
The main objective of our mediator class is to connect the Orders form with the ModifyStore and AddStore form, so when a qr store image is being modified or created, they send a message to the mediator so it sends the warning to the Orders table to update to show all the store with the corresponding values.

The other connection of the mediator is between the binnacle and all the actions phases, so when a new action is registered, as well as the start of a deliver, the creation of a new store, the selection for a new store order, etc, these classes sends the message through the mediator so it warns the binnacle that a new operation occurred and it keeps stored in the previously selected methods.

Last, we use the mediator to access to the trucks information into the delivery phase, so when we select some amount of trucks the mediator asks the singleton if all of those trucks are available, returning a boolean within some functions used during this process. Use of the singleton will be explained further.

Singleton

The well known singleton, in this project appears again as a great solution to the problem of keeping information between classes, where forms need to keep information between all of them.



Singleton here it's used more like an access point between some clases, like the mediator, to have access to some important variables or forms that are well used further.

If we take the example of the mediator, using a singleton to store the mediator helps us to have one instance of the clase mediator, and this way this class does not have to initialize every time we need, helping us to save time and a more complex way to implement due to the need of references to the classes that mediator need to regulate.

Indeed, singleton works as the mediator point of access to most of the classes the mediator needs to regulate.

Into the singleton we have also stored singles instances of our factories, to create these products when we need to.

As well, singleton stores information as a list of trucks, with all the 9 trucks we are using into the delivery route, as well as we don't have two initialize a truck every time we need to deliver a product, but to keep the trucks stored in a well known point of access for when those are required.

Last, but not least, singleton provides a single instance of the binnacle, and this is very important for us to keep the information stored in a single implementacion of the binnacle, managing to store the info even if we need to close the form to keep tracking the delivery. This way the complexity of the problem of storing the binnacle keeps reduced to store this single implementacion of it into the singleton and keep it for when we need to access it.