FIRST REPORT:

In the first exercise of this assignment, we used as main pattern the state pattern, which is a design pattern that allows an object to change its behavior based on its state. In our code, the Order class uses the state pattern to change its behavior in response to different actions that are performed on it, such as adding or removing products, managing products, paying and canceling or completing the order.

For example, when the addProduct method is called on an Order object, the behavior of the object will depend on its current state. If the object is in the ShoppingState state, then the product will be added to the shopping cart. If the object is in the CheckoutState state, then an exception will be thrown, because products cannot be added to the shopping cart once the order has been checked out.

The code consists of several classes that represent different states that the Order object can be in. For example, the Cancel class represents the state in which an order has been cancelled, and the CheckoutState class represents the state in which we only can modify our cart. Each of these classes implements the OrderState interface, which defines the behavior of an Order object in a particular state.

The SOLID principles, DRY, KISS, and YAGNI are principles that can be used to guide the design and implementation of our code.

DRY principle says that "Don't Repeat Yourself." In our code we avoid the repeated code in order to make the code more maintainable and comprehensible, because using the state pattern we have the implementation of each method a single time in each class, for example, methods for adding and removing products are only implemented in the ShoppingState, method for managing the cart is only implemented in the checkout state and so on.

KISS principle says that "Keep it simple, stupid." The code is as simple as possible making it more readable, this is fulfilled because each state has a simple task to perform, and since the tasks are different for each state they only worry about dealing with the cart situation at that moment, for example in CheckoutState, we only worry about managing the products that are in the current cart, not if we want to add a new product because it is supposed to be solved in the previous state.

YAGNI principle says that "You aren't gonna need it." This means that you should not implement features when you don't need them, as it may never be used. In the code there are not these features in order to simplify the code. This can be seen in our code because all the methods that we implemented are used to manage the cart operations, such as in the PaymentState, we implement additional methods for checking the validity of a given date to decide

if the order can be cancelled or have to be completed automatically, but all of them are used to keep moving to the next state.

SOLID principles have several principles to make software designs more maintainable and scalable. The ones that we used are:

Single Responsibility Principle (SRP): A class should have only one reason to change and a single responsibility entirely encapsulated by the class, this can be seen in our code because each state is in charge of doing only one thing, the ShoppingState let add/remove products, the checkoutState let manage the products, the paymentState let cancel/continue with the order, and so on.

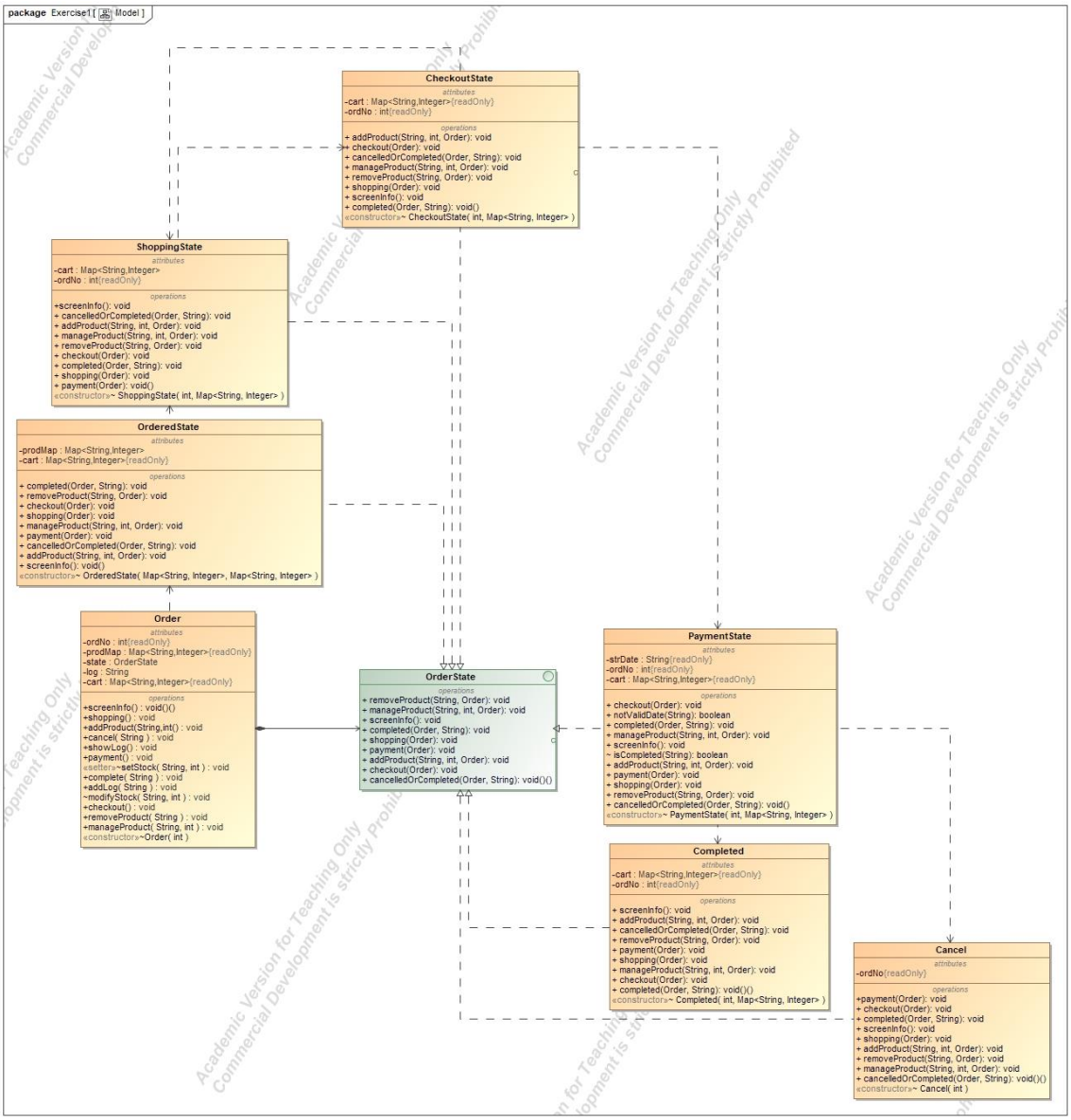
And the unique reason to change is a change of state.

Dependency Inversion Principle: High-level modules should not depend on low-level modules; both should depend on abstractions, this is fulfilled by our code because the states of the shopping cart depend on the OrderState interface

As a conclusion, our code is an example of implementation of the State pattern, and follows some design principles that will make the code easy to be maintained and extended.

The corresponding diagrams are the following ones:

-Class diagram:



-State diagram: