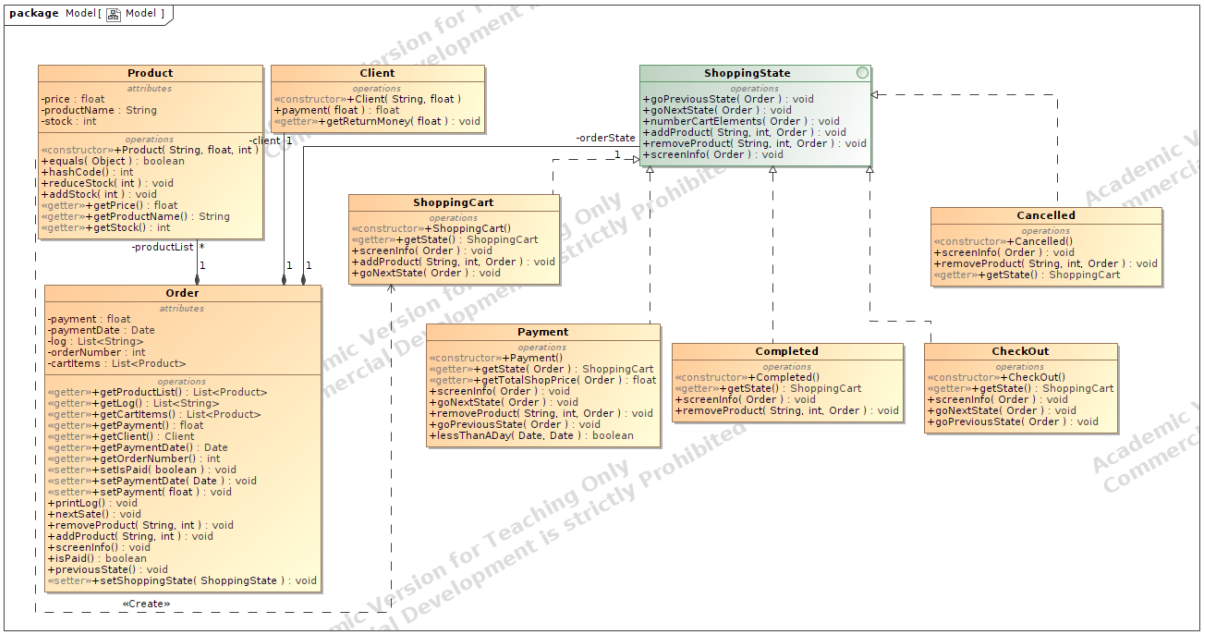
Marcos Seco Anderson & Diego Dopazo Garcia

E1 REPORT: Online shopping system

In this exercise we created an online shopping system. This online shop should have 5 different states. To do this we implemented the different states of the shopping process with 5 different classes: ShoppingCart, CheckOut, Payment, Completed and Cancelled.

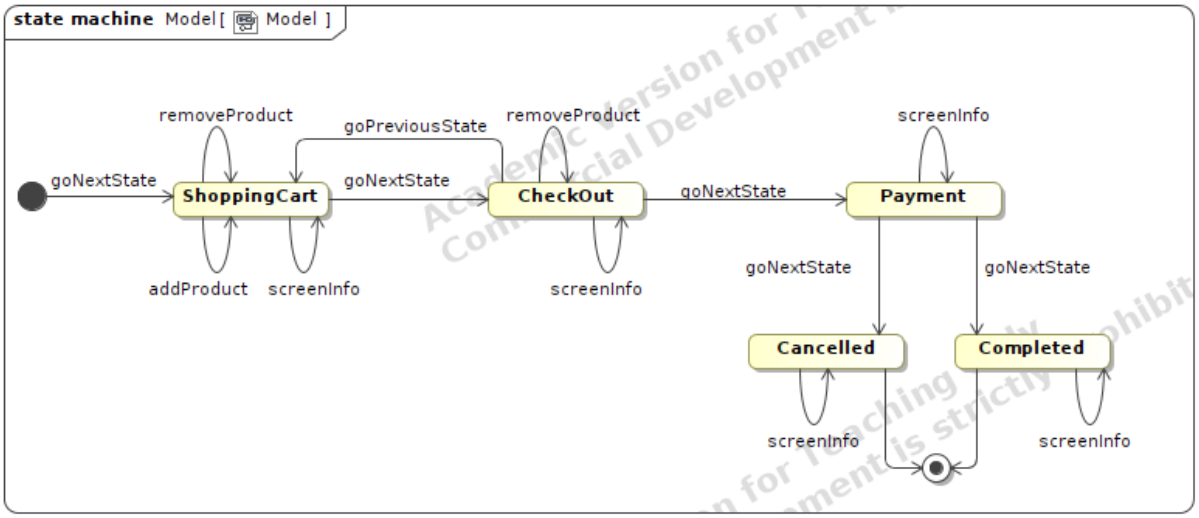
The class diagram looks like these:



This exercise follows the **State Pattern**, the state pattern is a behavioral design pattern that allows an object to alter its behavior when its internal state changes. The object will appear to change its class.

At the start, you may add items to the Shopping Cart as long as there is enough stock available for the chosen product. You can also remove added items from the cart. The moment you want to finalize the purchase you can do a check-out to finish the shopping. Then you move to the Checkout state, where at this point the client can go back to the previous step and keep managing the cart. Alternatively, and in order to avoid confusion, the system lets you remove the items from the previous step. After choosing the products and their quantities, the client pays for the order in the Payment state. Finally, after the payment, it is either accepted or canceled, and the state goes to either Cancelled or Completed.

In the implementation of this exercise we also decided to implement the **Singleton pattern**, this pattern ensures that a class has only one instance, and provides a global access point to that instance. In this case it is applied to the states. To do this the constructors are declared private, and a static getter (getState) , method allows access to the unique instance.



SOLID PRINCIPLES

Some examples of solid principles in this code include:

1. Single Responsibility Principle (SRP): Each class has a single, well-defined responsibility. For example, the Cancelled class is only responsible for handling the cancelled state of an order, and the Payment class is only responsible for handling the payment process.
2. Open/Closed Principle (OCP): The code is open for extension (by adding new states or functionality) but closed for modification (existing code is not modified). This is achieved by using an interface (ShoppingState) and implementing it in different classes.
3. Liskov Substitution Principle (LSP): Subtypes must be able to be used as a substitute for their parent type. This is achieved by implementing the ShoppingState interface in all classes.
4. Interface Segregation Principle (ISP): Clients should not be forced to depend on interfaces they do not use. This is achieved by having different methods in the ShoppingState interface that are implemented only by the classes that need them.
5. Dependency Inversion Principle (DIP): High-level modules should not depend on low-level modules. Both should depend on abstractions. This is achieved by using the ShoppingState interface and injecting it into the Order class.