**Adapter**

A customer have the ability to pay using credit or debit, so the system will include third party libraries that the client will interact with and handle those payment. Since our system doesn’t know how the user will be interacting with the interface supplied by the third party library, an Adapter class should be used to control interfaces where the developer can’t control them.

In our scenario, when a user wants to pay through one of the payment methods available, he will interact (interface) with the Adapter class that sits between the payment class (developed by us) and the third party libraries that those payment methods support.

**Facade + Adapter**

Assuming that we have a complete system, say the customer will check and buy an item, we will have two subsystems included, buy subsystem and check subsystem. Since we are working with n-tier system that is made of interface, business and data layers, we need to break down our whole system into smaller subsystem. Using Façade design pattern will allow us to develop cleaner code and readable, where on the client side, the whole system will be easier to use.

Another benefit of the Facade in our situation is to allow the use to interact and see what he needs to see, rather than seeing the whole system specially the business layer.