SOLID

# **Single Responsibility Principle**

“A class should have only one reason to change”

To ensure that the first principle of SOLID “Single Responsibility Principle” Two new Services were created “*AccountService*” and *“AccountDataStoreService*” to split up the actions of the **MakePayment** method.

The purpose of “*AccountService*” is to validate the payment depending on the varying allowed payment schemes and also update the balance of the account.

The purpose of *“AccountDataStoreService*” is to update the data store with the latest version of the account.

# **Open Closed Principle**

“A class should be open for extension but closed for modification”

No changes were made to ensure this principle was achieved. If the code was to be extended in the future then the Interfaces such as IMakePaymentRequest, IMakePaymentResult and IAccount should be used as the base class with new Classes extending their functionality.

For example there was a need to have a DelayedMakePayementRequest class which includes a property to delay the payment for a given time based on the PaymentDate property of the IMakePaymentRequest interface.

# **Liskov Substitution Principle**

“You should be able to change an instance using a subtype and your code should still work”

As is the case with the “Open Closed Principle” there are no classes that inherit from any other so no changes are required.

# **Interface Segregation Principle**

“Make fine grained interfaces that are client-specific. Clients should not be forced to implement interfaces they do not use.”

No changes were required to achieve this principle as there were no classes inheriting from other classes.

If in the future individual classes were necessary for the different Accounts of each Allowed Payment scheme then it would be wise to have a base interface IAccount with all of the necessary properties that are evident in each of the account types and create other interfaces for properties that are not shared thus removing the need for the implementation of unnecessary properties.

# **Dependency Inversion Principle**

“One should depend on abstraction and not concrete instances”

In order to achieve the principle of “Dependency Inversion” Interfaces are created for the classes to prevent high level modules relying on low level ones.

A static factory object class was created that allows for the centralised creation of new instances in order to have a single point of instantiation instead of newing up instances throughout the code.