**Credit Card Problem**

**Problem Statement:** We need to figure card type on basis of card number and create the instance of appropriate credit card class. So there are two problems basically one is to figure out what kind of card a specific record about and other one is how to create the appropriate object. One can be solve by using behaviour design pattern and other can be use by creational pattern.

**So Primary problem is to create instance of Card type class.**

**Design Pattern :** This problem can be solved by using Factory Pattern which is a Creational design pattern. Implementation is below :

using System;

/// <summary>

///Author : Yuri Narang

///Date : 10 July 2020

/// </summary>

namespace DesignPattern

{

// Product

abstract class CreditCard

{

public string cardNumber;

public DateTime expiration;

public string name;

}

// Concrete product

class VisaCC : CreditCard

{

public static bool isValidate(string number)

{

if (number.Length != 13 && number.Length != 16)

return false;

return (number[0] == '4');

}

}

// Concrete product

class MasterCC : CreditCard

{

public static bool isValidate(string number)

{

if (number.Length != 16)

return false;

return (number[0] == '5'

&& number[1] >= 1 && number[1] <= 5);

}

}

// Concrete product

class AmExCC : CreditCard

{

public static bool isValidate(string number)

{

if (number.Length != 15)

return false;

return (number[0] == '3'

&& (number[1] == 4 || number[1] == 7));

}

}

// Concrete product -

class Discover : CreditCard

{

public static bool isValidate(string number)

{

if (number.Length != 16)

return false;

return (number.Substring(0, 3) == "6011");

}

}

/// <summary>

/// Factory class

/// </summary>

abstract class CardFactory

{

public abstract CreditCard createCard(String number);

}

/// <summary>

/// Concrete factory class

/// </summary>

class CreditCardFactory : CardFactory

{

public override CreditCard createCard(String number)

{

CreditCard cc = null;

VisaCC vqq = new VisaCC();

if (number.Length == 0 || number.Length > 19)

return cc;

else if (MasterCC.isValidate(number))

cc = new MasterCC();

else if (VisaCC.isValidate(number))

cc = new VisaCC();

else if (AmExCC.isValidate(number))

cc = new AmExCC();

else if (Discover.isValidate(number))

cc = new Discover();

return cc;

}

}

}

**What is achieved using design pattern(s).**

The Factory Method Design Pattern is used, when we need to create the object (i.e. instance of the Product class) without exposing the object creation logic to the client. To achieve this, in the factory method design pattern we will create an abstract class as the Factory class which will create and return the instance of the product, but it will let the subclasses decide which class to instantiate.

In this example client will call createCard() of creditCardFactory class by passing card number as parameter. In return he gets object of card type class without knowing object creation logic. In this way we have achive loose coupling between classes.

We can add new card type at any stage later without disturbing existing clients or logic.