**Single Responsibility Principle C# Example**

“A Class should have only one reason to change”

A class should have only a single responsibility (i.e. only one potential change in the software's specification should be able to affect the specification of the class)

It states that:

Each software module should have one and only one reason to change.

Or,

There should never be more than one reason for a class to change.

Make sure each of your methods does only one thing.

A class should have one and only one reason to change, meaning that a class should have only one job.

Let's go through the problem first. Have a look at the code given below:

public class BankAccount  
   {  
       public BankAccount()  
       {  
       }  
       public string AccountNumber { get; set; }  
       public decimal AccountBalance { get; set; }  
       public decimal CalculateInterest()  
       {  
           //Code to calculate Interest  
       }  
   }

Its Model class for object ***BankAccount***, and it's also saving the ***Calculating Interest*** for the bank account. Now look at the few change Request we received from business:

1.       Please add a new Property ***AccountHolderName*** .

2.       Some new rule has been introduced to calculate interest.

These 2 are totally different type of change request. One is changing on features; where as other one is impacting the functionality. So that's SRP Principle says, we should have 2 different type of reason to change one class. Most of the times we keep functions in our domain classes, which contain business rules. This violates **Single Responsibility Principle**.

Now let's go through how we can achieve this thing by implementing **Single Responsibility Principle**.

Look at the code below:

public interface IBankAccount  
   {  
       string AccountNumber { get; set; }  
       decimal AccountBalance { get; set; }  
   }  
   public class BankAccount : IBankAccount  
   {  
       public string AccountNumber { get; set; }  
       public decimal AccountBalance { get; set; }  
   }

Now our class BankAccount is just responsible for properties of the BankAccount. If we have add any new Business Rule in Calculation of Interest, we don't need to change BankAccount class.

Now about saving the data, we will create another class. Let's say **InterstCalculator**, as below:

public interface IInterstCalculator  
   {  
       decimal CalculateInterest();  
   }  
   public class InterstCalculator : IInterstCalculator  
   {  
       public decimal CalculateInterest(IBankAccount account)  
       {  
           // Write your logic here  
           return 1000;  
       }  
   }

InterestCalculator class requires no changes, in case we need to add a new Property AccountHolderName.  So that's correct implementation of ***Single Responsibility Principle.*** We have also used Interfaces to communicate between InterestCalculator and BankAccount classes. That will help us to manage dependencies between classes.

Diagram example:  


