Introduction

In object oriented programming systems, we have lots of concepts which we have to think of when designing systems. A few of them are very important and a few of them are considered pillars of object oriented programming systems. Here I start with the first pillar of object oriented programming systems and that is encapsulation.

Definition

Encapsulation is the first pillar or principle of object-oriented programming. In simple words, “Encapsulation is a process of binding data members (variables, properties) and member functions (methods) into a single unit”. And Class is the best example of encapsulation.

Important points

* Through encapsulation a class can hide the internal details of how an object does something. Encapsulation solves the problem at the implementation level.
* A class or structure can specify how accessible each of its members (variables, properties, and methods) is to code outside of the class or structure. Encapsulation simplifies the interaction between objects. An object can use another object without knowing all its data or how its data is maintained. For example, a Client object might have name, address, company, and department properties. If a Bank object wants to use a Client object, it can request the name and address for the bank without needing to know the company and department details of the Client object.
* With the help of encapsulation, a class can change the internal implementation without hurting the overall functionality of the system.
* Encapsulation protects abstraction.

Need or purpose of encapsulation

* To hide and prevent code (data) from the outside world (here the world means other classes and assemblies).
* To prevent code (data) from accidental corruption due to programming errors so that we can deliver expected output. Due to programming mistakes, code may not behave properly and it has an effect on data and then it will affect the functionality of the system. With encapsulation we can make variables, properties, and methods private so it is not accessible to all but accessible through proper channels only to protect it from accidental corruption from other classes.
* To have a class better control over its fields (validating values etc…).

Ways to achieve encapsulation with code example

We can achieve encapsulation by the following ways. Take a look at the methods to achieve encapsulation with code example:

1. By using the get and set methods (Accessors and Mutators)

Hide   Copy Code

public class Account

{

private string accoutName;

*// get methods*

public string GetAccount()

{

return accoutName;

}

*// Set method*

public void SetAccount(string name)

{

accoutName = name;

}

}

static void Main()

{

string name ="SAVING\_ACCOUNT";

Account account = new Account();

account.SetAccount(name);

name = string.Empty;

name = account.GetAccount();

}

In the above example we use the get and set methods (GetAccount and SetAccount) to return account and set account name. We use

the private variable accountName and as it is not accessible directly, to use this variable, we use the get and set methods.

1. By using properties (read only properties, write only properties)

Like the above example we can achieve encapsulation using properties also.

We can use a property (which has a get and set part), or we can use a read only property (which has only a get part) or we can also use a write only property (which has only a set part). But in all cases we can achieve encapsulation.

Have a look at the following example using properties.

Hide   Shrink http://www.codeproject.com/images/arrow-up-16.png   Copy Code

*// Encapsulation using properties*

public class Account

{

private string accoutName="SAVING\_ACCOUNT";

*// property which has get and set*

public string AccoutName

{

get

{

return accoutName;

}

set

{

accoutName = value;

}

}

private string address="India";

*// readonly property*

public string Address

{

get

{

return address;

}

}

private string phone = "1234567890";

*// writeonly property*

public string Phone

{

set

{

phone=value;

}

}

}

static void Main()

{

*// Encapsulation using properties*

string name = string.Empty;

Account account = new Account();

*// call get part*

name = account.AccoutName;

*// change the value*

name = "CURRENT\_ACCOUNT";

*// call set part*

account.AccoutName = name;

string address = string.Empty;

*// call readonly property*

address = account.Address;

*// now address has value “India”*

string phone = "1234567890";

*// call writeonly property*

account.Phone = phone;

*// now account.Phone has value “1234567890”*

}

Here when we create a new instance of the account class, all the private variables in the account class (account name, address, and phone) are assigned with values. In the main class we can skip the variables (name, address, and phone) and directly use System.Console to write the output. I

use the variables so that during debugging we can check how the values of the variables in the main class change after every call to the properties.