# Encapsulation and concealment of information

## Do my Coding

In the design area there are two concepts that are often used together

encapsulation (encapsulation) and information hiding (information hiding).

The concept of encapsulation is usually used in the context of OO languages and means hiding the data (*).Open changes (**) Data violate encapsulation, since now any class client can change the inner state of the object without the knowledge of the very class.It can break some invariants, i.e.The conditions for which the author calculated the author and is formally or informally, its customers should rely.

(*)

Sometimes the concept of encapsulation is applied in a broader sense.For example, it is said that the factory encapsulates information about the specific type of object being created.In this context, encapsulation is synonymous with information hiding.

(**)

Although it is believed that the class is unmomitable in generally to be open data, but from a practical point of view, an open unchangeable field though disrupts encapsulation, usually does not lead to the same serious accompaniment problems as open changeable data.

As a result, when designing a class / module, two components appear: open part

interface and closed part

Implementation.In this case, the class or module interface should not simply duplicate the closed part through the acsessors (properties or GET / SET methods), but it should give the client a higher-level, abstract interface.In other words, the open part of the class should mostly talk about what makes this class or module, and hide unnecessary details from customers, to hide, as it does.

Abstraction and Icapsulation complement each other: abstraction is directed to the observed behavior of the object, and encapsulation is engaged in the internal device.
Castle Boch, Object-oriented Analysis and Design

It should be intuitive that the hiding fields / data is only a particular case of information, which means that the concept of information hiding this should not be limited.

Here is a small and naive example:

```
Public Class Employee {}
Public Class Paystub.
{

Private Readonly List <Employee> _employees = new list <employee> ();

Public List <Employee> Employees => Employees;

Public Decimal ComputePayroll ()

{

// Using _employees

Return 42;

}
}

Public Class Paystub2.
{

Private Readonly List <Employee> _employees = new list <employee> ();

Public Void AddEmployee (Employee E) {_Employees.add (E);}

Public Decimal ComputePayroll ()

{

// Using _employees

Return 42;

}
}
```

In both cases, we have no open data, but the design quality is clearly different.Both classes have encapsulation, but the first implementation hides the details of the implementation not completely.

Here is another example, which shows that concealing information

This is more than closed data.

Suppose you are developing the Yantrpreyz app, which makes something important.Any normal application usually requires some configuration: well, the connection parameters to the database or server, and other valuable information.And so, the motley artector together with an equally mate client is asked to read the configuration from the configuration file.

In the process of a conversation with them, you understand that no one really knows why it is necessary to read the configuration from the file, which format should be in and what exactly should be stored there.

Now you become before choosing: You can unwind the configuration information to a smooth layer throughout the application. Each component to which some parameters need is climbing in the App Config, will pull out the necessary data from there, the XML or JSON is passing and will be ready to serve. On the other hand, it is obvious that the decision on exactly where the configuration is stored and in what format may change in the future. Therefore, a more sane solution will hide the location information and configuration format in one module, for example, using Configuration and ConfigurationProvider classes. In this case, when (yes, it is when, and not if) the requirements will change, then only the implementation of the CONFIGURATIONPROVIDER class will be changed, and all other users of this class or configuration will remain unchanged. Similarly, when the format is changed, only the parsing process will also be changed, and not consumers configuration.

This example seems contrived, but it is not so!We are quite often faced with the variability of requirements, but we use, unfortunately, one of the two approaches:

Completely ignore the possibility of changing the requirements and do everything in the forehead or

Create a super challenging solution, with a dozen noise levels, which should withstand changes in any direction without changing the code in general.

A more reasonable approach is somewhere in the middle.Every time I start developing some feature, I think how many pieces in the code will have to change if the requirements or parts of the implementation will change significantly.At the same time, I do not try to reduce the number of changes to 1 (well, such as if we follow the SRP, then there should be only one place, in case of change of claim).I try so that these places are not enough, and the changes were simple.

Encapsulation and concealment of information are designed to combat complexity and to simplify the development of the system.Abstraction and encapsulation allow you to think about the problem without going into unnecessary details: to calculate the amount of wages, I need to collect data about employees and the wage calculation class will make everything else.To properly use this class, I do not need to know or think about how it will do it.I only need to provide the necessary input data.

Hiding the parts also hurts limit the cascade of change.As soon as new requirements come and we

will have to change the algorithm for calculating wages, then we will have to change only one place, and not twenty.

Now we have enough examples to understand what concealing information is.

Hiding information

This is concealing design details that may change in the future to limit cascade changes in the software system and to simplify its development and maintenance.

Yes, it came out a little persecuted, but nothing.Hiding information

This is the principle of design that allows packaging certain knowledge to a box that can be accurately hidden in one place, and strain in the future or change its contents.