Encapsulation

In c#, **Encapsulation** is a process of binding the [**data members**](https://www.tutlane.com/tutorial/csharp/csharp-variables-with-examples)and [**member functions**](https://www.tutlane.com/tutorial/csharp/csharp-methods-functions-with-examples) into a **single unit**. In c#, the **class** is the **real-time example** for encapsulation because it will combine various types of [data members](https://www.tutlane.com/tutorial/csharp/csharp-variables-with-examples) and [member functions](https://www.tutlane.com/tutorial/csharp/csharp-methods-functions-with-examples) into a **single unit**.

Generally, in c# the encapsulation is used to **prevent alteration of code (data) accidentally** from the **outside** of [functions](https://www.tutlane.com/tutorial/csharp/csharp-methods-functions-with-examples). In c#, by defining the **class fields with**[**properties**](https://www.tutlane.com/tutorial/csharp/csharp-properties-get-set) we can protect the data from accidental corruption.

If **we define class fields with** [properties](https://www.tutlane.com/tutorial/csharp/csharp-properties-get-set), then the encapsulated [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples) **won’t allow us to access** the fields directly instead, we need to use **getter and setter** [functions](https://www.tutlane.com/tutorial/csharp/csharp-methods-functions-with-examples) to **read or write data** based on our **requirements**.

If you observe the above code, we defined [**variables**](https://www.tutlane.com/tutorial/csharp/csharp-variables-with-examples) with [**private access modifiers**](https://www.tutlane.com/tutorial/csharp/csharp-access-modifiers-public-private-protected-internal#divcspvtm)and exposing those variables in a **public** way by using [**properties**](https://www.tutlane.com/tutorial/csharp/csharp-properties-get-set) **get** and **set** accessors.

In case, if you want to make any **modifications** to the defined **variables**, then we can make it by using [**properties**](https://www.tutlane.com/tutorial/csharp/csharp-properties-get-set) with **get** and **set** accessors.

**C# Partial Class**

In c#, a **partial class** is useful to **split** the functionality of a particular class into **multiple** **class** files and all these files will be **combined** into one **single** class file when the **application is compiled.**

While working on large scale projects, multiple **developers** want to work on the same [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples) file at the same time. To solve this problem, c# provides an ability to **spread** the **functionality** of a particular [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples) into multiple [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples) files using partial [keyword](https://www.tutlane.com/tutorial/csharp/csharp-keywords-reserved-contextual).

In c#, we can use partial [keyword](https://www.tutlane.com/tutorial/csharp/csharp-keywords-reserved-contextual) to split the definition of a particular [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples), [structure](https://www.tutlane.com/tutorial/csharp/csharp-structures-structs), **interface** or a [method](https://www.tutlane.com/tutorial/csharp/csharp-methods-functions-with-examples) over two or more source files.

## Difference between Dictionary and Hashtable in C#

* [Hashtable](https://www.tutlane.com/tutorial/csharp/csharp-hashtable) is a non-generic type of collection so it can store elements of different data types but Dictionary is a generic type of collection so it can store elements of the same data type.
* [Hashtable](https://www.tutlane.com/tutorial/csharp/csharp-hashtable) is available with **System.Collections** [namespace](https://www.tutlane.com/tutorial/csharp/csharp-namespaces-with-examples) but the dictionary is available with **System.Collections.Generic** [namespace](https://www.tutlane.com/tutorial/csharp/csharp-namespaces-with-examples).
* In the [foreach](https://www.tutlane.com/tutorial/csharp/csharp-foreach-loop-with-examples" \o "C# Foreach Loop with Examples" \t "_blank) loop, you need to use **DictionaryEntry** property to get the key/value pair from hashtable but we need to use **KeyValuePair** property to access key/value pair elements from dictionary.
* When compared with **dictionary** object, the [hashtable](https://www.tutlane.com/tutorial/csharp/csharp-hashtable" \o "C# HashTable with Examples" \t "_blank) will provide a lower performance because the [hashtable](https://www.tutlane.com/tutorial/csharp/csharp-hashtable" \o "C# HashTable with Examples" \t "_blank) elements are of object type so the boxing and unboxing process will occur when we store or retrieve values from the [hashtable](https://www.tutlane.com/tutorial/csharp/csharp-hashtable" \o "C# HashTable with Examples" \t "_blank).
* In c#, the [hashtable](https://www.tutlane.com/tutorial/csharp/csharp-hashtable" \o "C# HashTable with Examples" \t "_blank) will throw an error if we try to find a key that does not exist but the dictionary object will return null in case the defined key not exists.