C#

1. **Boxing and Unboxing**

* **Boxing:**

is the process of converting data from a value to a reference type(**tham chiếu)**. For example, the Boxing process will initialize(**khởi tạo**) an object and copy the value to a refference type called Boxing

* **Unboxing**: cho phép chuyển đổi(**allows convert**) từ đối tượng sang bất kỳ loại giá trị nào**(value type**).

1. **Class and Struct**

Class and Struct are exactly the same, except for some problems:

* **Struct** is a value type(kiểu giá trị) and **class** is a reference type(kiểu tham chiếu).
* Because struct is a value type, we can declare without using the new operator.
* Struct does not support inheritance
* Struct don’t have destructor, constructor must have parameter

1. **Encapsulation (Tính đóng gói)**

Encapsulation is the process of packaging one or more items inside a package

Encapsulation include **Public Access Specifier, Private Access Specifier, Protected Access Specifier.**

* **Public Access Specifier**: allows any public member to be accessed from outside that class.
* **Private Access Specifier**: allows a class to hide member variables and its member functions, only functions in the same class can access private members
* **Protected Access Specifier** allows a subclass to access member variables and member functions of base class.

<https://vietjack.com/csharp/tinh_dong_goi_trong_csharp.jsp>

1. **Struct**

The struct statement is like a class, which defines a new data type, with more than one member of your program.

1. **Class**

Class is a data type which containing properties, methods used to build the executable(thực thi) program.

1. **Enum**

* enum in C # is a keyword used to declare an enumeration(sự liệt kê).
* The default starts at 0.

1. **Tính kế thừa (Inheritance)**

Inheritance is one of the four aspects of object-oriented programming(OOP), the ability to allow us to define a new class based on an existing class, inheritance makes extending code - maintenance easier.

1. **Tính đa hình (Polymorphism)**

**Polymorphism** Allows an operation to have different behaviors on different objects, it including Overloading, Overriding

**Overload** allows us to change parameters in the constructor

**Overriding** allows us to reuse method from base class, We need to use the **virtual** keyword with the base class method (parent class) and the **override** keyword with the subclass class method

 Sự khác nhau giữa override và overload là gì? <http://minhhn.com/lap-trinh-c-sharp/su-khac-nhau-giua-override-va-overload-trong-csharp/>

<http://minhhn.com/lap-trinh-c-sharp/tinh-da-hinh-polymorphism-trong-csharp/>

1. **Exception**

**Exception** is the problem - an error occurred during program execution. when a program is running and having a error, so the program ends immediately.

**try - catch**

1. **Interface**

**Interface** is a set of only declarations without a definition,These components may be: Method, Property, Event, Indexers

1. **Interface vs Abstract**

|  |  |  |
| --- | --- | --- |
|  | Interface | Abstract |
| - Override: | Not use | use |
| - Constructor, destructor: | Have | don’t have |
| Speed |  | faster |
|  | Contains only declarations, no content | contains properties and methods |

1. **File I/O**

The FileStream class in C # helps in reading, writing and closing files. This class inherits from the abstract class Stream.

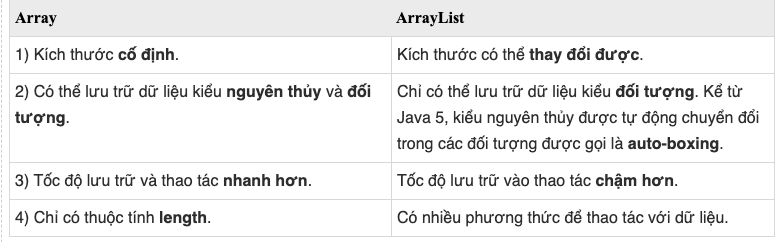
1. **Delegate**

Delegate to define a representation for functions which have the same function type.

The **delegate** is used to implement(thực hiện) events and call-back methods

For example, delegates are used for plus and minus functions, because they have the same parameter,and return the same data type.

1. **Array và ArrayList**

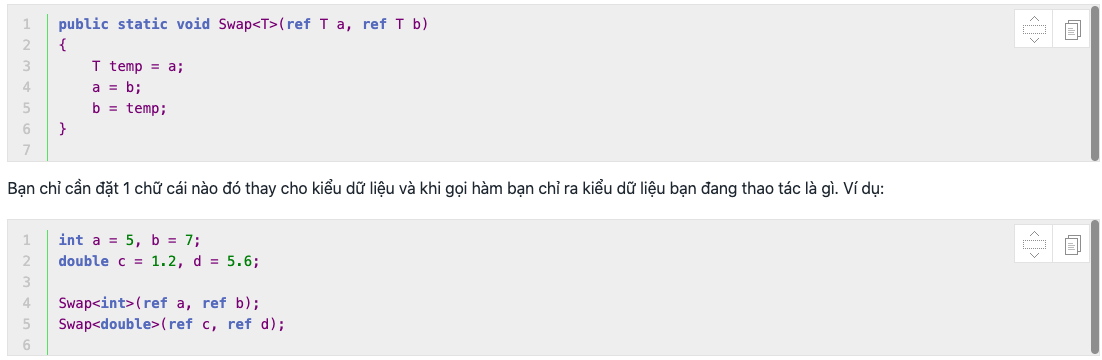


1. **Event**

**Event** is the class that notifies to another classes. The event is built on Delegate

1. **Generic**

**Generic** was born to help us reduce code and increase reuse.



1. **Ref and Out parameter**

**Ref and out** are two keywords used to pass a reference to a function.

1. **SortedList**

**SortedList** is also a Collections store data as Key - Value. The key represents a key like the index of the array and the Value is the corresponding value of that key.

We can access elements in SortedList through Key (like **Hashtable**) or via element index (like **ArrayList**).

SortedList is a combination of ArrayList and Hashtable.

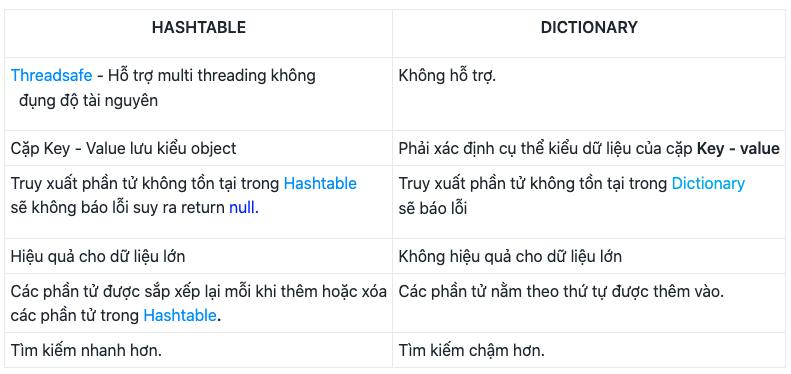
|  |  |
| --- | --- |
| GetByIndex(int Index) | Trả về giá trị **Value**tại vị trí **Index**trong SortedList. |
| GetKey(int Index) | Trả về giá trị **Key**tại vị trí **Index**trong SortedList. |

1. **Hashtable**

The same with SortedList. But Hashtable just can access element by key

1. **Dictionary**

The same with Hashtable. But have many difference



ádasdasd