BÀI 4:
HƯỚNG ĐỐI TƯỢNG TRONG C#

Thời gian: 180 phút

Giảng viên: PHẠM PHÚ KHƯƠNG

Email: phamphukhuong@dtu.edu.vn

Điện thoại:0905635421

DUYTAN edu.vn

Nội Dung

- Giới thiệu lịch sử phát triển
- Đối tượng
- Lớp
- Các thành phần của lớp

Video tham khảo:

https://www.youtube.com/watch?v=z10ncAjTzC4

LỊCH SỬ PHÁT TRIỂN

LỊCH SỬ PHÁT TRIỂN

- **Đầu tiên là: Lập trình truyền thống trải qua hai giai đoạn**
  - Giai đoạn sơ khai, khi khái niệm lập trình mới ra đời: là lập trình tuyến tính
  - Tiếp theo: lập trình cấu trúc

LẬP TRÌNH TUYẾN TÍNH

- Là phương pháp xuất hiện đầu tiên. Phương pháp này đơn giản chỉ là viết tất cả mã lệnh vào một hàm main duy nhất và chạy
- **Ví dụ**

```
static void Main(string[] args)
{
    double tong = 0;
    int n;
    Console.WriteLine("Nhap vao so n");
    n = int.Parse(Console.ReadLine());
    for (int i = 1; i <=n; i++)
    {
        tong = tong + Math.Pow(i, 2);
    }
    Console.WriteLine(" Tong S la" + tong);
    Console.ReadLine();
}
```

LẬP TRÌNH CẤU TRÚC

- Chia chương trình lớn ra thành các chức năng, mỗi chức năng được đưa vào 1 hàm. Khi cần dùng đến chức năng nào thì ta sẽ gọi hàm tương ứng.
- Mỗi chương trình con lại có thể chia nhỏ ra nữa.
- Hầu hết các ngôn ngữ lập trình đều hỗ trợ phương pháp này.

Ưu điểm:

- Chương trình được module hoá, dễ hiểu, dễ bảo trì.
- Dễ dàng tạo ra các thư viện phần mềm.

Nhược điểm:

- Dữ liệu và xử lý tách rời.
- Khi cấu trúc dữ liệu thay đổi sẽ dẫn đến thuật toán bị thay đổi.
- Không tự động khởi tạo, giải phóng dữ liệu động.
- Không mô tả được đầy đủ, trung thực hệ thống trong thực tế.

- **Ý nghĩa:** Với mong muốn xây dựng một phương pháp lập trình trực quan, mô tả trung thực hệ thống trong thực tế vì thế phương pháp lập trình hướng đối tượng ra đời..
- **Khái Niệm:**
  - Phương pháp hướng đối tượng bằng cách tạo những đối tượng cụ thể, tập hợp các thuộc tính của đối tượng.
  - Sau đó nhóm các đối tượng thành một nhóm, loại bỏ các thuộc tính quá cá biệt, chỉ giữ lại các thuộc tính chung nhất, nhóm thành từng lớp.

THẢO LUẬN

1. Hãy nêu những ngôn ngữ hướng đối tượng mà các em đã được học.
2. Một ngôn ngữ được gọi là hướng đối tượng khi đảm bảo những đặt trưng nào

THẢO LUẬN

❖ **Câu hỏi:**

1. Hãy nêu những ngôn ngữ hướng đối tượng mà các em đã được học.

Trả lời: Java, C++.

THẢO LUẬN

❖Câu hỏi:

2. Một ngôn ngữ được gọi là hướng đối tượng khi đảm bảo những đặt trưng nào

Trả lời: đảm bảo 3 đặt trưng đó là :

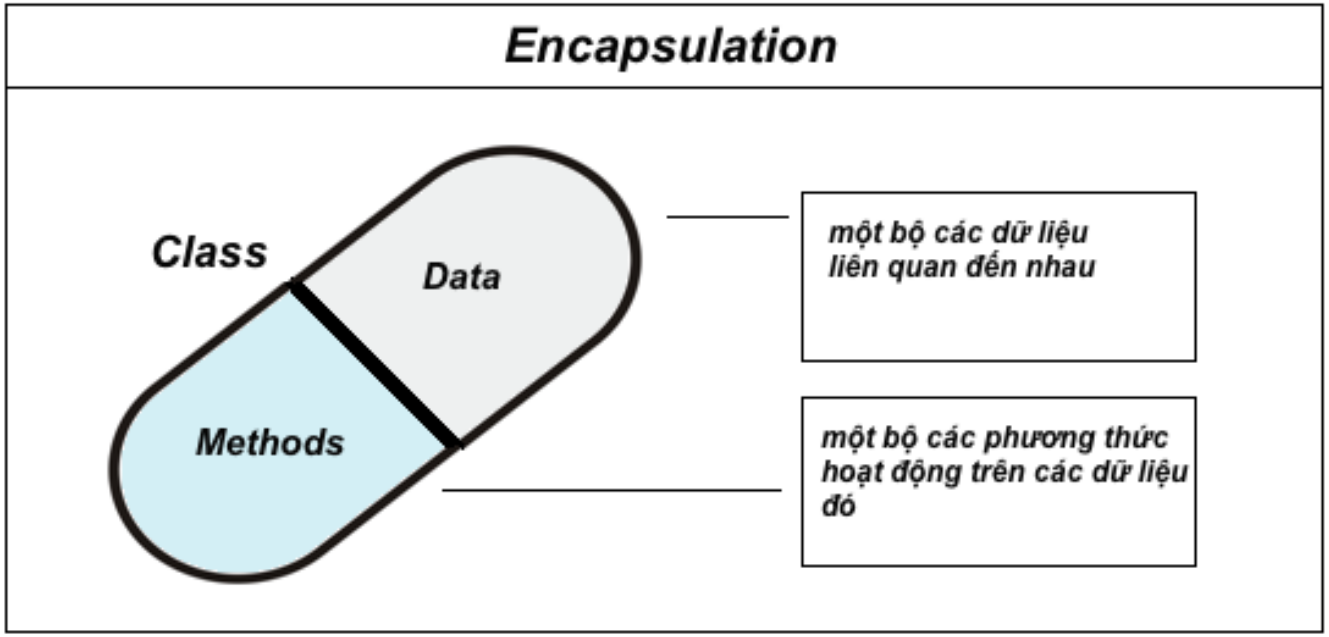
- *Tính đóng gói (encapsulation)*
- *Tính kế thừa (inheritance)*
- *Tính đa hình (polymorphism)*

TÍNH ĐÓNG GÓI (Encapsulation)

- Mục đích: hạn chế sự truy cập tự do vào dữ liệu, không quản lý được.
- Là một cơ chế dùng một vỏ bọc kết hợp thành phần dữ liệu và các thao tác trên dữ liệu đó thành một thể thống nhất, tạo nên sự an toàn, tránh việc sử dụng không đúng thiết kế…
- Việc truy nhập đến dữ liệu phải thông qua các phương thức của đối tượng lớp.

TÍNH ĐÓNG GÓI (Encapsulation)

- Các dữ liệu và phương thức có liên quan với nhau được đóng gói thành các lớp để tiện cho việc quản lý và sử dụng. Điều này được thể hiện qua cách ta xây dựng 1 class.



TÍNH ĐÓNG GÓI (Encapsulation)

- Đóng gói còn để che giấu một số thông tin và chi tiết cài đặt nội bộ để bên ngoài không thể nhìn thấy. Cụ thể:
  - **Các biến** thường sẽ có phạm vi là private.
  - **Các phương thức** thường sẽ có phạm vi là public.
  - **Sinh viên hãy giải thích vì sao?**

TÍNH ĐÓNG GÓI (Encapsulation)

- **Các biến** thường sẽ có phạm vi là private. Vì đây chính là các thông tin nội bộ của lớp không thể để truy cập 1 cách tuỳ tiện được (che giấu thông tin).
- **Các phương thức** thường sẽ có phạm vi là public. Vì đây chính là các hành vi (thao tác) mà lớp hỗ trợ cho chúng ta thực hiện những công việc nhất định nên cần phải cho phép mọi người có thể sử dụng được.

- **Tính kế thừa (inheritance):** là khả năng xây dựng các lớp mới từ những lớp đã có
  - Các lớp có thể kế thừa nhau để tận dụng các thuộc tính, các phương thức của nhau, ngoài ra có thể bổ sung các thuộc tính và phương thức riêng của mình.
  - Có thể sử dụng lại mã nguồn, tiết kiệm tài nguyên.

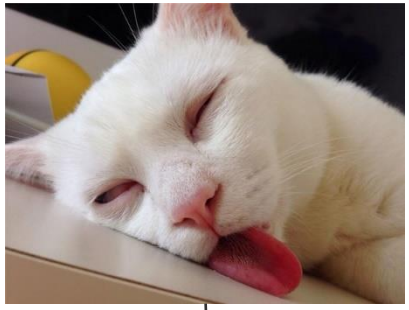
- **Tính đa hình (polymorphism):** là nhiều hình thái, hình thức và nhiều kiểu tồn tại. Mỗi phương thức có cách thể hiện khác nhau trên nhiều loại đối tượng khác nhau.
  - Thể hiện ở nạp chồng phương thức, viết chồng phương thức, lớp trừu tượng, giao diện.

- **Ưu Điểm:**
  - Không còn nguy cơ dữ liệu bị thay đổi tự do trong chương trình.
  - Khi thay đổi cấu trúc dữ liệu của một đối tượng, không cần thay đổi mã nguồn của đối tượng khác, mà chỉ thây đổi đối tượng đó mà thôi
  - Có thể sử dụng lại mã nguồn
  - Phù hợp với dự án lớn
- **Nhược điểm:**
  - Hơi phức tạp
  - Phải tư duy và mô tả được thế giới thực

LỚP VÀ ĐỐI TƯỢNG

ĐỐI TƯỢNG

- Đối tượng: là một chủ thể bất kỳ nào đó
- Ví dụ:



Kiểu đối tượng

Con Mèo

Ô Tô

Điện Thoại

ĐỐI TƯỢNG (tt)

- Đối tượng trong thế giới thực bao gồm thuộc tính và các hành vi

Ví dụ:

Thuộc tính chính là những thông tin, đặc điểm của đối tượng. Ví dụ: một người sẽ có họ tên, ngày sinh, màu da, kiểu tóc, . .

Hành vi là những thao tác, hành động mà đối tượng đó có thể thực hiện. Ví dụ: một người sẽ có thể thực hiện hành động nói, đi, ăn, uống, . . .

ĐỐI TƯỢNG (tt)

Đối tượng trong phần mềm cũng tương tự như các đối tượng trong thế giới thực.

- Một đối tượng lưu trữ thuộc tính của nó trong biến
- Thể hiện các hành vi của nó ra bên ngoài thông qua các phương thức

CÂU HỎI

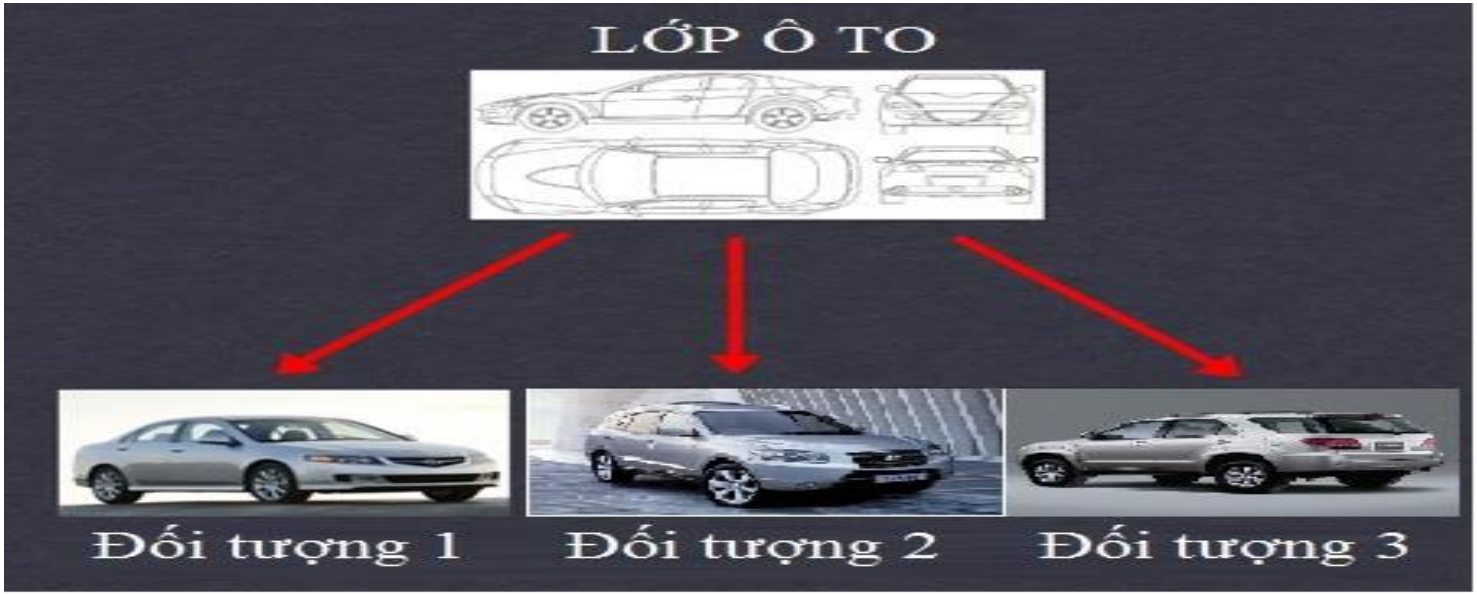
- **Giả sử chúng ta cần làm một phần mềm về quản lý sinh viên của một trường đại học.**
  1. Sinh viên hãy cho biết các đối tượng trong phần mền cụ thể là gì
  2. Các thuộc tính của đối tượng đó

CÂU TRẢ LỜI

- **Giả sử chúng ta cần làm một phần mềm về quản lý sinh viên của một trường đại học.**
  1. Các đối tượng trong phần mềm cụ thể là: Giáo viên, Sinh viên,….
  2. Lấy ví dụ đối tượng là sinh viên thì thuộc tính là: họ tên, MSSV, ngày sinh, địa chỉ, điện thoại……

LỚP

- Lớp là một *khuôn mẫu* định nghĩa *thuộc tính* và *phương thức* chung cho tất cả các đối tượng thuộc cùng một loại
- Ví dụ:



- Một đối tượng là một *thể hiện* của lớp, thuộc về một lớp.
-

KHAI BÁO LỚP

```
[Phạm vi truy cập] class <Tên lớp> [:lớp cơ sở][:interfaces…]
{
}
```

- Phạm vi truy cập: định nghĩa khả năng truy cập của lớp đó, sử dụng một trong các từ khóa: Public, Internal, Protected, Protected Internal, Private
- class: từ khóa đánh dấu bắt đầu bằng một lớp
- Tên lớp: Tên của một lớp
- Lớp cơ sở: là lớp cha mà lớp hiện tại có thể thừa kế. Một lớp chỉ có thể thừa kế từ một lớp cha
- Interfaces: là giao diện mà lớp này có thể thừa kế. Một lớp có thể thừa kế nhiều interfaces

KHAI BÁO LỚP (tt)

Từ khóa	Phạm vi truy cập
Public	Truy cập ở bất cứ nơi đâu
Internal	Có thể được truy cập bởi các bất cứ đâu trong project hiện tại, nhưng không thể được truy cập bởi các project tham chiếu đến nó
Protected	Chỉ có thể được truy cập trong cùng một lớp hoặc trong những lớp được kế thừa từ lớp này
Protected Internal	Truy cập trong chương trình hiện tại (lắp ráp) hoặc bất kỳ lớp nào có kế thừa từ lớp này
Private	Chỉ có thể được truy cập trong cùng một lớp

KHAI BÁO LỚP (tt)

Phạm vi truy cập là public

Tên lớp là SinhVien

```
public class SinhVien
{
        //Thân của lớp
}
```

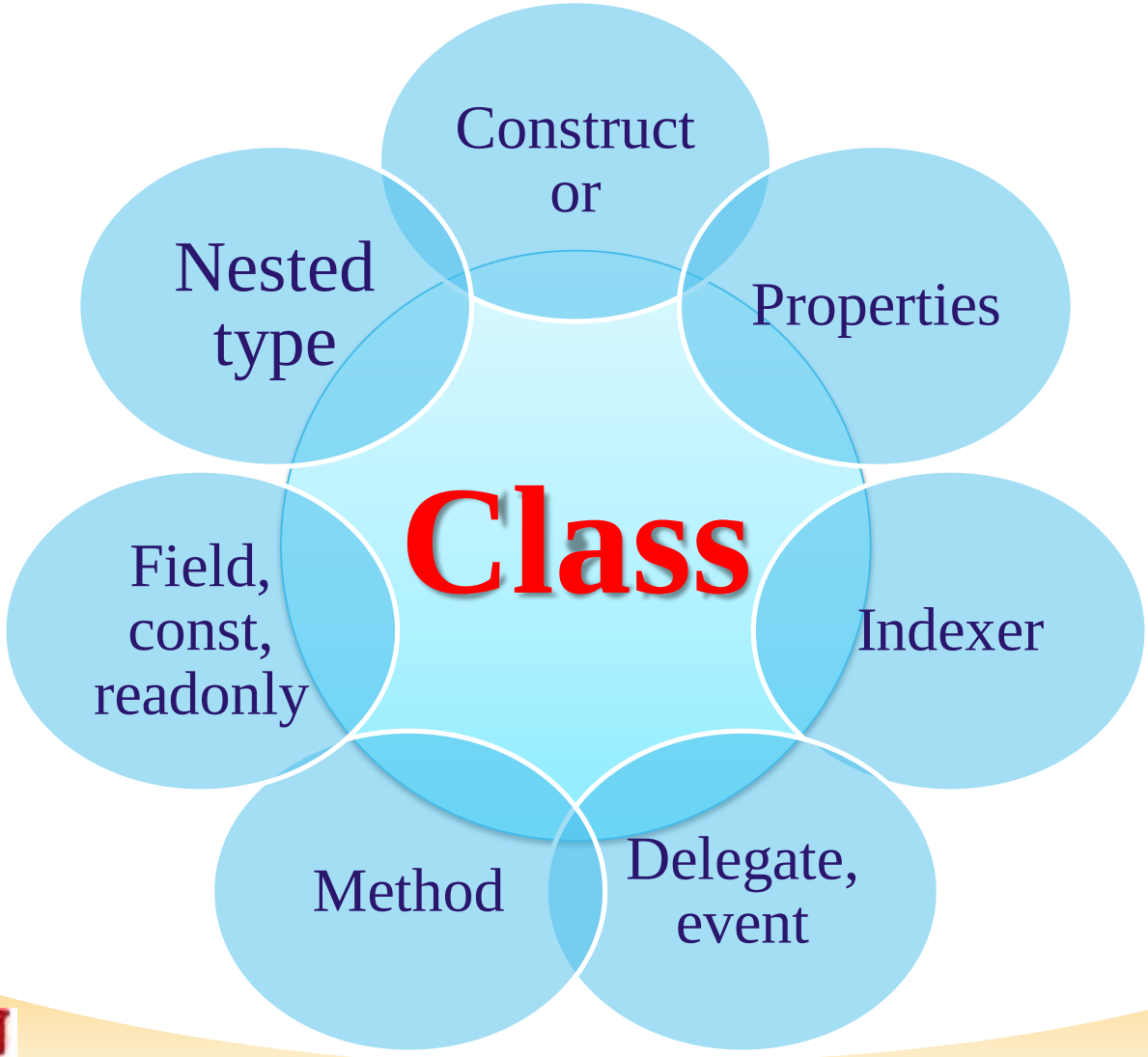
Không có lớp cơ sở

Mặc định xuất phát từ System.Object

THẢO LUẬN

- **Vậy lớp có trước hay đối tượng có trước**

CÁC THÀNH VIÊN CỦA LỚP



CÁC BIẾN THÀNH VIÊN

Các biến thành viên của lớp thể hiện các thuộc tính của lớp đó và được khai báo bên trong lớp như sau

```
class <tên lớp>
{
   // khai báo những biến của lớp
   [Phạm vi truy cập] <kiểu dữ liệu>
   <tên biến>;
   // …
}
```

CÁC BIẾN THÀNH VIÊN (tt)

Ví dụ:

```
class SinhVien
  {
    //Khai bao bien thanh vien
     public string hoten;
     private int tuoi;
     float diemtoan, diemvan;
  }
```

PROPERTY

Property cung cấp một cơ chế linh hoạt để lấy, gán giá trị, hoặc tính toán cho các giá trị của các biến thành viên ở có phạm vi truy cập là private.

```
public <kiểu> <tên thuộc tính>
{
    get {return <giá trị thuộc tính>;}
    set {<xử lý biến value>;}
}
```

Chú ý:

- Mã trong get sẽ chạy khi thuộc tính được truy xuất
- Mã trong set sẽ chạy khi thuộc tính được gán giá trị
- Phạm vi truy cập của thuộc tính luôn luôn phải được khai báo là public

PROPERTY

Ví dụ:

```
class SinhVien
  {
   //Khai bao thuoc tinh
     public int Tuoi
     {
        get { return tuoi; }
        set { tuoi = value; }
     }
  }
```

PHƯƠNG THỨC

Phương thức là khối lệnh thực hiện các chức năng, các hành vi xử lý của lớp và được khai báo như sau:

```
[Phạm vi truy cập] <kiểu trả về> <Tên phương thức>
([danh sách đối số])
{
    <khối lệnh>;
}
```

Trong đó:

- Kiểu trả về: có thể là kiểu **void**, kiểu cơ sở hay một lớp.
- Tên phương thức: đặt theo qui ước giống tên biến.
- Danh sách thông số: có thể rỗng

PHƯƠNG THỨC (tt)

Ví dụ:

```
class SinhVien
  {
    //Khai bao phuong thuc
    public float DiemTb()
      {
         return ( diemtoan+ diemvan)/2;
      }
  }
```

HÀM KHỞI TẠO (CONSTRUCTOR)

Hàm khởi tạo cũng là một phương thức của lớp (nhưng khá đặc biệt) dùng để tạo dựng một đối tượng mới.

```
[Phạm vi truy cập] <Tên hàm khởi tạo> ([danh sách đối số])
{
        //Khối lệnh
}
```

Lưu ý:

- Bắt buộc phải cùng tên với tên lớp
- Không khai báo kiểu
- Không có kết quả trả về.

HÀM KHỞI TẠO (CONSTRUCTOR)

- Hàm khởi tạo mặc định (nếu không khai báo)
  - Không có tham số
  - Được sử dụng khi tạo đối tượng không có thông tin gì
  - Vd lớp SinhVien có hàm khởi tạo mặc định là SinhVien()

HÀM KHỞI TẠO (CONSTRUCTOR)

- Hàm khởi tạo do người dùng định nghĩa
  - Có thể không chứa tham số hoặc chứa một hoặc nhiều tham số
  - Dùng để tạo đối tượng với một vài thông tin ban đầu

```
public SinhVien(string shoten, int stuoi)
    {
        hoten = shoten;
        tuoi = stuoi;
    }
```

HÀM KHỞI TẠO (CONSTRUCTOR)

- Hàm khởi tạo private là hàm khởi tạo đặc biệt của lớp. Nó ngăn cản khởi tạo đối tượng từ lớp này

Lớp SinhVien này có hàm khởi tạo với thuộc tính truy cập là private.

Ở hàm Main, đoạn code khởi tạo đối tượng sv từ lớp SinhVien sẽ bị lỗi khi chương trình biên dịch.

LỖI

```
public class SinhVien
  {
    //...
    private SinhVien()
    {
    }
  }
  class Program
  {
    static void Main(string[] args)
    {
      SinhVien sv = new SinhVien();
    }
  }
```

HÀM HỦY (DESTRUCTOR)

Hàm hủy là hàm đặc biệt dùng để làm sạch bộ nhớ

- Chỉ có một hàm hủy duy nhất
- Hàm hủy không thể được kế thừa hoặc nạp chồng
- Hàm hủy được gọi một cách tự động. Hàm hủy được khai báo như hàm khởi tạo nhưng không có thuộc tính truy cập và tham số truyền vào và bắt đầu bằng dấu "~"

```
public class SinhVien
   {
      //...
      ~SinhVien()//Hàm Hủy
      {
      //Thực thi câu lệnh
      }
   }
```

BÀI TẬP

❖ Tạo Lớp Xe:

- Bao gồm các trường: Biển số (string), Tên xe (string), Trọng tải (int), Ngày đăng kiểm (DateTime), Tiêu chuẩn bằng (int)
- Phương thức Nhập() void, và Xuất() void

CÂU HỎI

1. Định nghĩa Lớp (class)?
2. Biến instance là gì?
3. Bạn có thể nói gì về Contructor?
4. Mục đích của Constructor mặc định?
5. Khối static là gì?

YOUTUBE

1. https://www.youtube.com/watch?v=GAvhe6oe-_4
2. https://www.youtube.com/watch?v=8ARnYQgShY8&list=PLxefhmF0pcPl919wrtpzbx4tjpXAtPQ6e