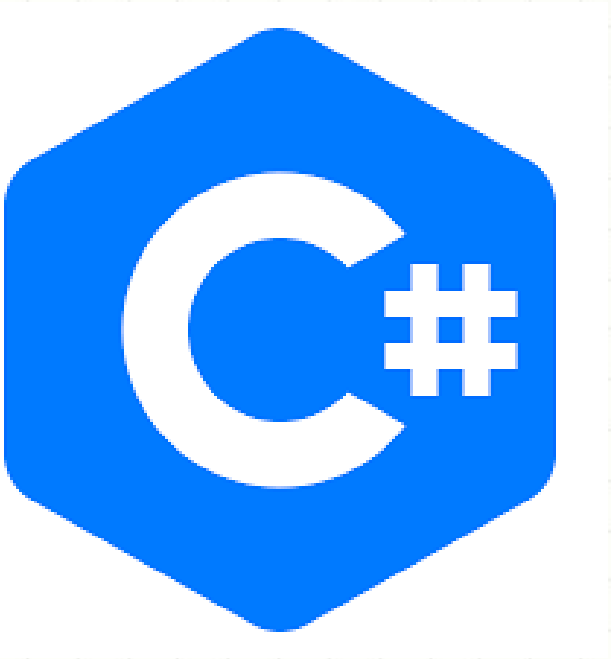
Tổng Quan Về C#

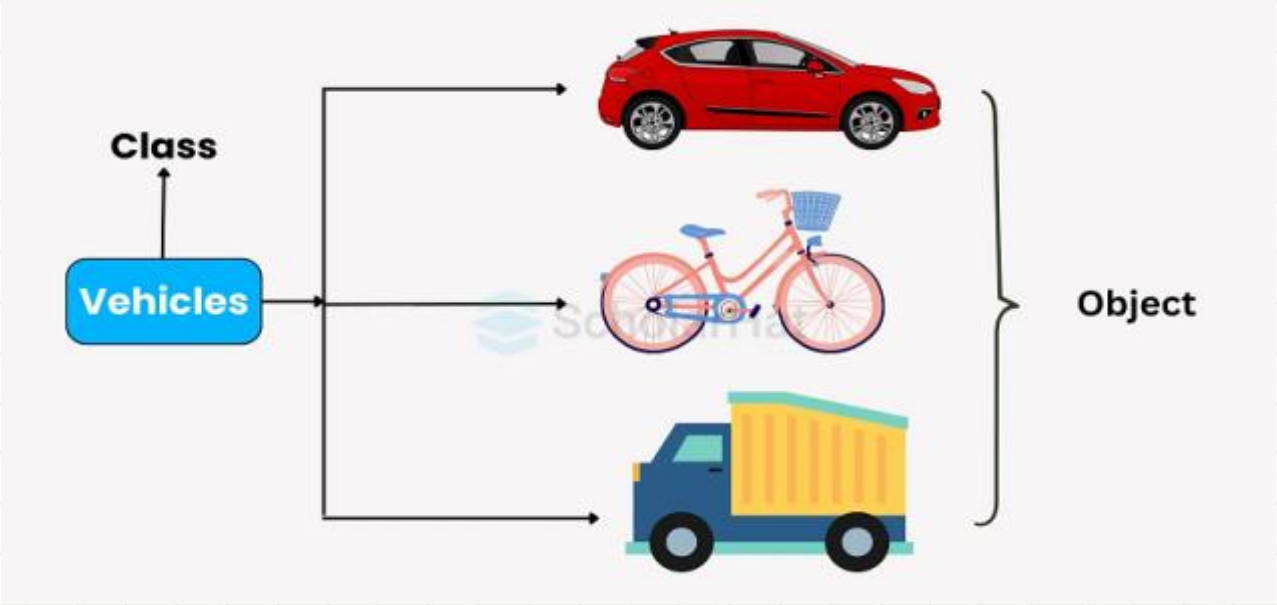


Nội dung

1	Cú pháp khai báo lớp
2	Khai báo và tạo lập đối tượng
3	Phương thức Thiết lập – Constructor
4	Phương thức Phá hủy – Destructor
5	Thành viên tĩnh – static member

Cú pháp khai báo lớp

Lớp là một mô tả trừu tượng của nhóm các đối tượng cùng bản chất, ngược lại mỗi một đối tượng là một thể hiện cụ thể cho những mô tả trừu tượng đó. Lớp là bản thiết kế và lập trình. Đối tượng là cái tạo ra trong lúc chạy chương trình



Cú pháp khai báo lớp

- Một lớp bao gồm các thành phần dữ liệu (thuộc tính) và các hàm thành phần (phương thức)
- Lớp trong C# thực chất là một kiểu dữ liệu do người sử dụng định nghĩa.
- Trong C#, dùng từ khóa class để khai báo một lớp

Cú pháp khai báo lớp

Để xây dựng lớp, ta cần:

- Xác định các thuộc tính (dữ liệu)
  - Những gì mà ta biết về đối tượng – giống như một struct
- Xác định các phương thức (hành vi)
  - Những gì mà đối tượng có thể làm
- Xác định các quyền truy cập

Cú pháp khai báo lớp

```
class <ClassName> {
    // Khai báo các thuộc tính và phương thức
}
```

Cú pháp khai báo lớp

- Khi khai báo thành phần (thuộc tính, phương thức) của lớp, ta cần chỉ định thêm bổ từ truy cập (access modifier): private, public, protected, internal
- Những thành phần private chỉ có thể được truy cập trong nội bộ lớp
- Những thành phần public có thể được truy cập cả từ bên trong và bên ngoài lớp
- Những thành phần protected chỉ có thể được truy cập trong nội bộ lớp và trong lớp con dẫn xuất của nó
- Những thành phần internal chỉ có thể được truy cập trong cùng assembly (project)

Cú pháp khai báo lớp

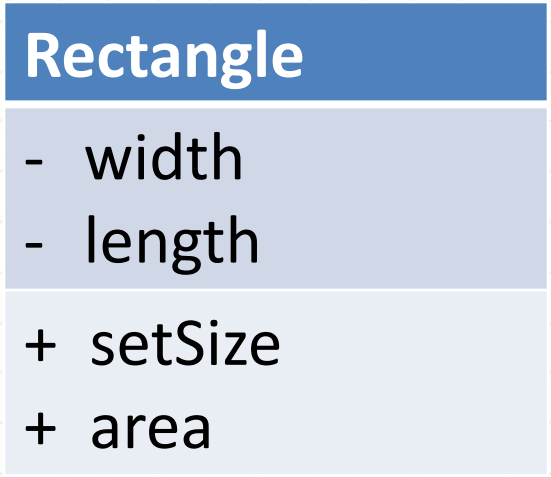
Modifier	Trong cùng class	Class con (kế thừa)	Cùng assembly	Assembly khác
public	✔	✔	✔	✔
private	✔	✘	✘	✘
protected	✔	✔	✘	✘
internal	✔	✔	✔	✘
protected internal	✔	✔	✔	✔ (nếu là class con)
private protected	✔	✔ (cùng assembly)	✔ (cùng assembly)	✘

Cú pháp khai báo lớp

- Thông thường, các thuộc tính sẽ thuộc diện private, các phương thức sẽ thuộc diện public
- Nếu không chỉ rõ, mặc định phạm vi truy cập của các thành phần sẽ là private
- Khi khai báo lớp, ta cũng có thể chỉ định phạm vi truy cập cho lớp. Nếu không chỉ rõ, mặc đinh phạm vi truy cập của lớp là internal

Cú pháp khai báo lớp

Khi dùng sơ đồ UML để mô tả lớp, các thành phần public sẽ có dấu (+), private sẽ có dấu (-), protected sẽ có dấu (#) và internal sẽ có dấu (~) đứng trước tên



Cú pháp khai báo lớp

Ví dụ một khai báo lớp:

```
class Rectangle {
    private int width;
    private int length;
    public void setSize (int w, int l);
    public int area();
}
```

Cú pháp khai báo lớp

- Thuộc tính: Các thuộc tính được khai báo giống như khai báo biến
- Phương thức: Các phương thức được khai báo giống như khai báo hàm và phải được định nghĩa thi hành (implementation) ngay trong lớp

Cú pháp khai báo lớp

```
class Rectangle {
    private int width;
    private int length;
    public void setSize (int w, int l) {
        width = w;
        length = l;
    }
    public int area() {
        return (width * length);
    }
}
```

Cú pháp khai báo lớp

```
class Rectangle {
private:
    int width, length;
public:
    void setSize (int w, int l);
    int area();
};
void Rectangle::setSize (int w, int l) {
     width = w;
     length = l;
}
int Rectangle::area() {
     return (width * length);
}
```

Định nghĩa thi hành bên ngoài lớp trong C++

Khai báo và tạo lập đối tượng

❖Cách 1:

```
ClassName ObjName = new ClassName();
```

❖Cách 2:

```
ClassName ObjName;
ObjName = new ClassName();
```

❖Cách 3:

```
var ObjName = new ClassName();
```

Khai báo và tạo lập đối tượng

```
class Rectangle {
   private int width;
   private int length;
   public void setSize(int w, int l) {
      width = w;
      length = l;
   }
   public int area() {
      return (width * length);
   }
}
```

```
class App {
    static void Main () {
        Rectangle rec1 = new
     Rectangle();
        Rectangle rec2;
        rec2 = new Rectangle();
        var rec3 = new Rectangle();

        rec1.setSize(10, 30);
        Console.WriteLine(rec1.area());
        rec2.setSize(15, 40);
        Console.WriteLine(rec2.area());
        rec3.setSize(20, 50);
        Console.WriteLine(rec3.area());
     }
}
```

Khai báo và tạo lập đối tượng

- Khi ta gọi một phương thức trên một đối tượng, C# ngầm truyền địa chỉ của đối tượng vào phương thức đó qua biến **this**.
- Nhờ **this**, bên trong phương thức, ta có thể:
  - Truy cập trực tiếp đến đối tượng hiện tại
  - Phân biệt biến thành viên và biến cục bộ hay tham số (nếu trùng tên)

Khai báo và tạo lập đối tượng

```
class Rectangle {
    private int width;
    private int length;
    public void setSize (int width, int length) {
        this.width = width;
        this.length = length;
    }
    public int area() {
        return (width * length);
    }
;
```

Phương thức thiết lập (constructor)

- Phương thức thiết lập (constructor) là một loại phương thức đặc biệt dùng để khởi tạo đối tượng của lớp.
- Bất kỳ một đối tượng nào khi được khai báo đều phải sử dụng một constructor để khởi tạo các giá trị thành phần (data) của đối tượng.
- Constructor phải được đặt tên trùng với tên lớp và không có giá trị trả về (kể cả void).
- Constructor phải có phạm vi truy cập public

Phương thức thiết lập (constructor)

- Một lớp có thể có nhiều constructor được định nghĩa chồng
- Các loại constructor:
  - Constructor mặc định
  - Constructor có tham số
  - Constructor sao chép

Phương thức thiết lập (constructor)

Constructor mặc định:

- Thiết lập các dữ liệu ban đầu cho đối tượng
- Nếu không khai báo bất kỳ một constructor nào cho lớp, C# sẽ tự tạo ra một constructor mặc định là một phương thức rỗng (không làm gì cả)

Phương thức thiết lập (constructor)

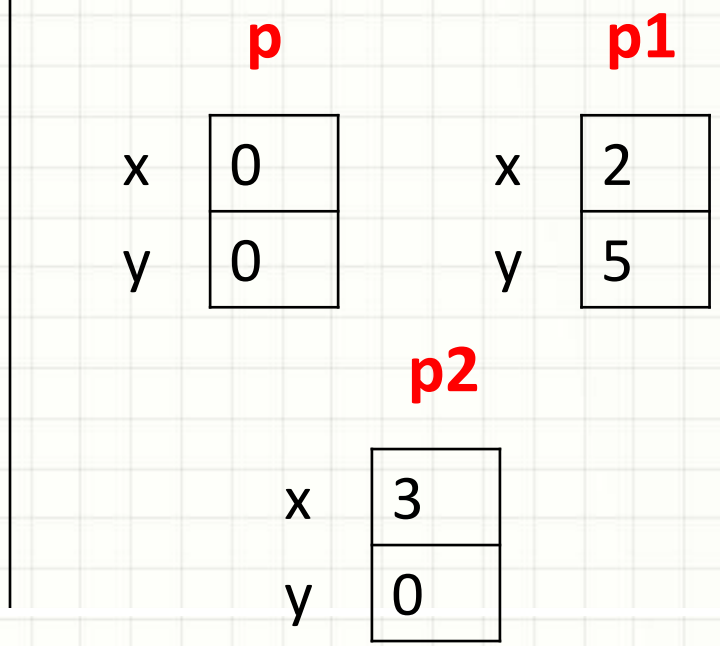
Constructor có tham số:

- Thiết lập các dữ liệu ban đầu cho đối tượng dựa vào tham số truyền vào
- Constructor có tham số có thể được dùng để ép kiểu dữ liệu của tham số về kiểu của lớp

Phương thức thiết lập (constructor)

```
class Point {
   private int x;
   private int y;
   public Point () {
       x = 0;
       y = 0;
   }
   public Point (int xx, int yy = 0) {
       x = xx;
       y = yy;
   }
   public void move(int dx, int dy){
       x += dx;
       y += dy;
   }
}
```

```
class App {
  static void Main () {
     Point p = new Point();
     Point p1 = new Point(2, 5);
     Point p2 = new Point(3);
  }
}
```



Phương thức thiết lập (constructor)

```
class PhanSo {
   private int tu;
   private int mau;
   public PhanSo (int i) {
       tu = i;
       mau = 1;
   }
   public PhanSo (int tt, int mm) {
       tu = tt;
       mau = mm;
   }
}
```

```
class App {
  static void Main () {
     PhanSo p = new PhanSo(); // lỗi
     PhanSo p1 = new PhanSo(3);
   }
}
```

Phương thức thiết lập (constructor)

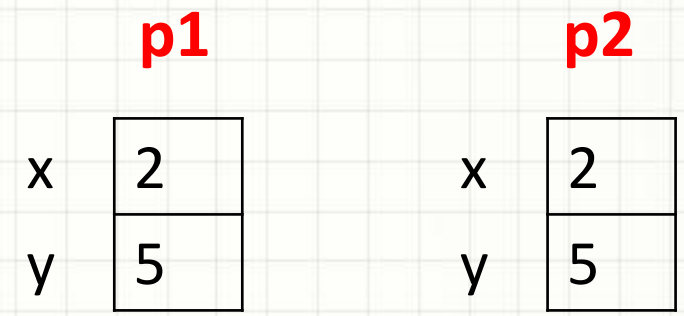
Constructor sao chép:

- Nhận tham số đầu vào là một đối tượng cùng lớp
- Constructor sao chép dùng để tạo ra đối tượng mới từ một đối tượng khác với "mức độ giống nhau" tùy theo cách người lập trình muốn

Phương thức thiết lập (constructor)

```
class Point {
   private int x;
   private int y;
   public Point (int xx, int yy = 0) {
       x = xx;
       y = yy;
   }
   public Point (Point p) {
       x = p.x;
       y = p.y;
   }
   public void move(int dx, int dy){
       x += dx;
       y += dy;
   }
}
```

```
class App {
  static void Main () {
     Point p1 = new Point(2, 5);
     Point p2 = new Point(p1);
  }
}
```



Tại sao ta không dùng lệnh gán Point p2 = p1 ?

Phương thức phá hủy (destructor)

- Phương thức phá hủy (destructor) được dùng để thu hồi, dọn dẹp tài nguyên cấp cho đối tượng khi đối tượng hết hoạt động
- Mỗi lớp chỉ có một destructor
- Destructor được đặt tên trùng với tên lớp và có dấu ~ phía trước, không giá trị trả về
- Destructor phải có phạm vi truy cập public
- Do C# có cơ chế Garbage Collection nên destructor không còn cần thiết nữa

Thành viên tĩnh của lớp

- Thành viên tĩnh của lớp là thành phần thuộc về cả lớp chứ không thuộc về một đối tượng cụ thể nào
- Thành viên tĩnh có thể là thuộc tính tĩnh hay phương thức tĩnh
- Để khai báo một thành viên tĩnh, ta dùng từ khóa static
- Phương thức tĩnh không có đối tượng this

Thành viên tĩnh của lớp

```
using System;
class Student {
    private string name;
    public static int count = 0;            // thuộc tính tĩnh
    public Student(string st) {
        name = st;
        count++; // mỗi khi tạo 1 student, tăng biến đếm
    }
    public void show() {
        Console.WriteLine(name);
    }
    // phương thức tĩnh
    public static void showCount() {
        Console.WriteLine($"Current number of students: {count}“);
    }
}
```

Thành viên tĩnh của lớp

```
class App {
    static void Main() {
        Student s1 = new Student("An");
        Student s2 = new Student("Binh");

        s1.show();
        s2.show();

        // gọi phương thức tĩnh qua tên lớp
        Student.showCount();  // Current number of students: 2
    }
}
```

Một số nội dung tự tìm hiểu

Object Initializer

Auto-implemented Property

Xử lý ngoại lệ (Exception Handling)