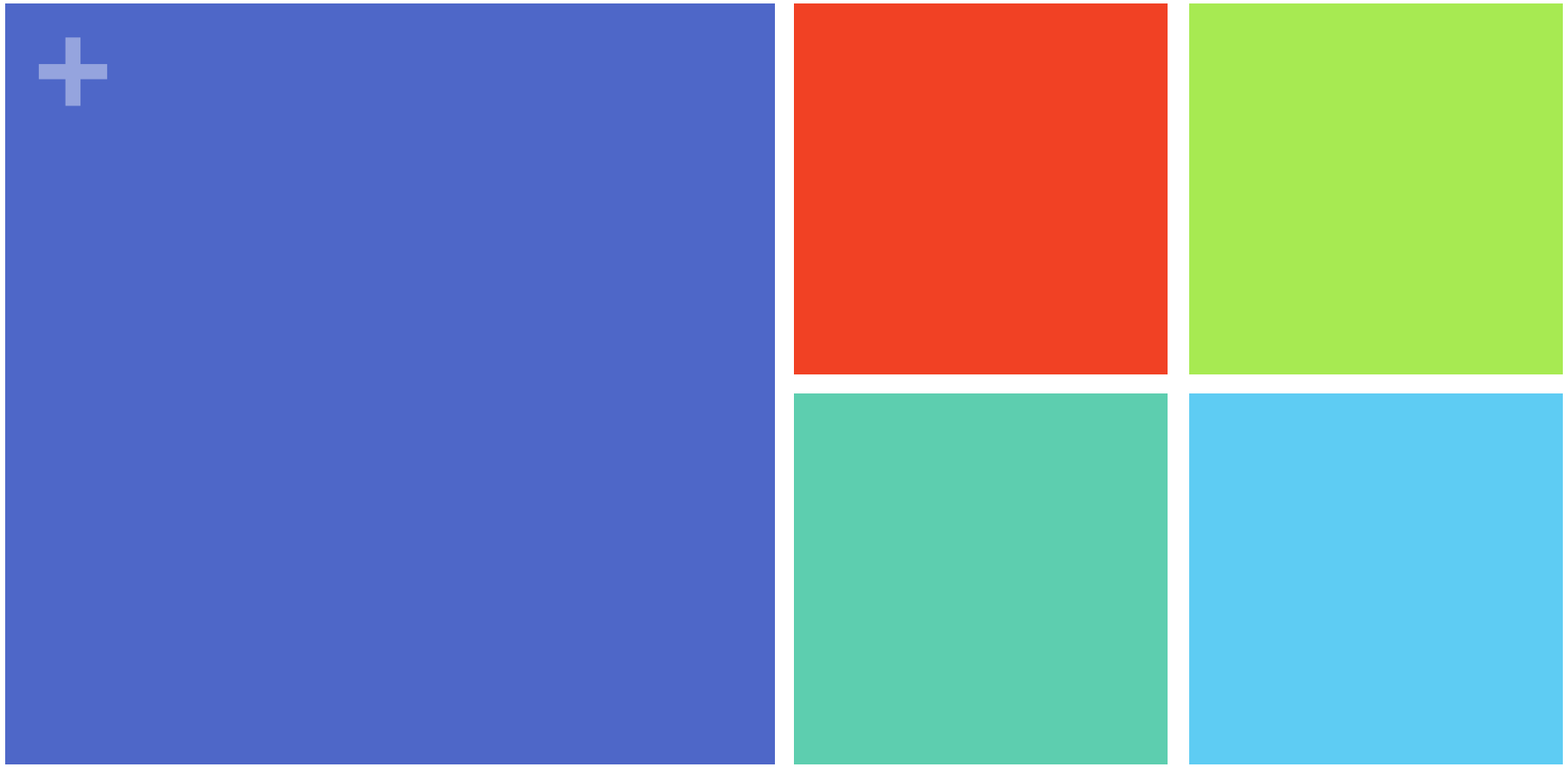


# THÀNH PHẦN CƠ BẢN

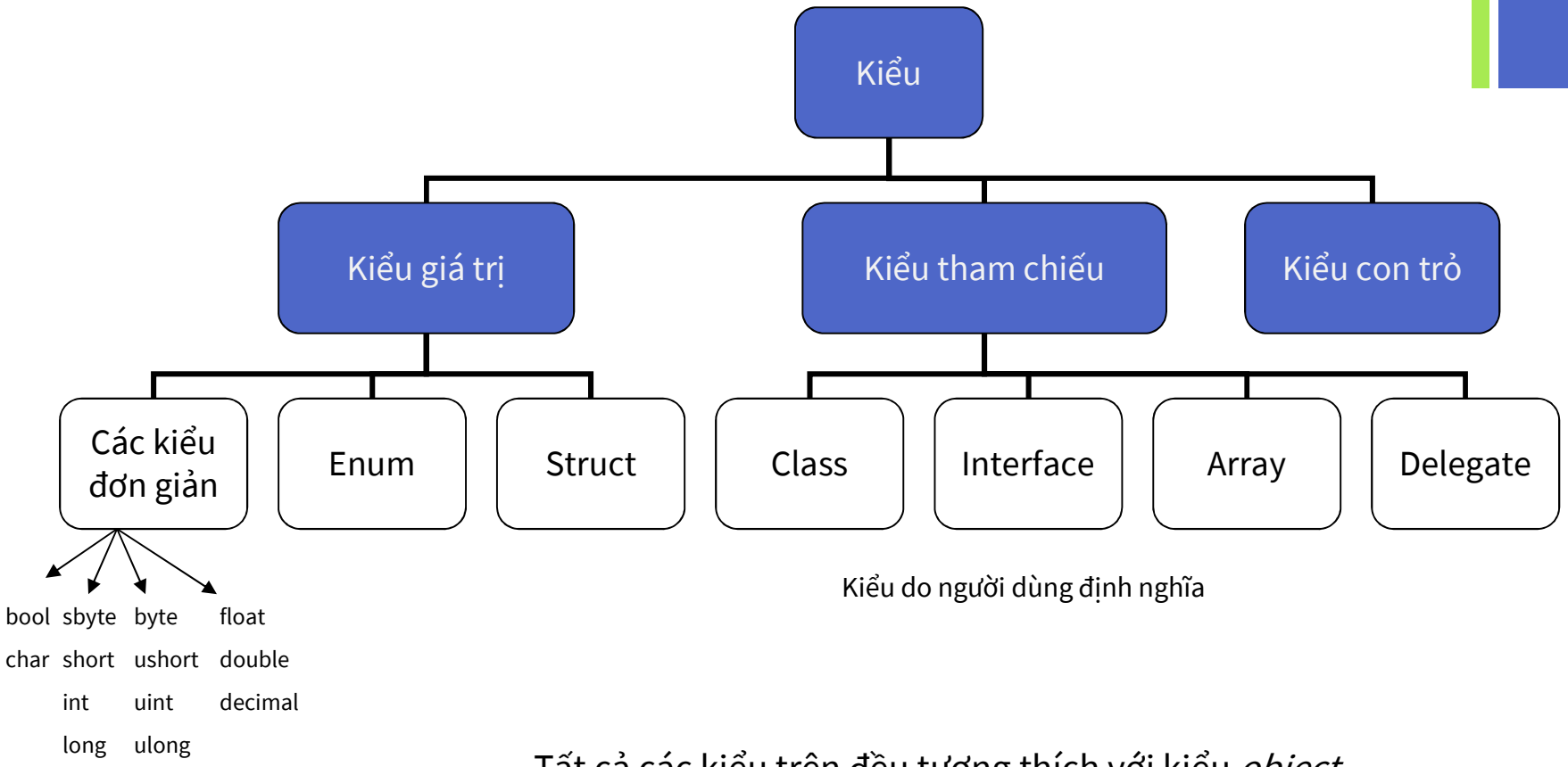
TS. ĐẶNG THÀNH TRUNG

# NỘI DUNG

- Các kiểu dữ liệu
- Toán tử và độ ưu tiên toán tử
- Khai báo biến

# KIỂU DỮ LIỆU VÀ TOÁN TỬ

# PHÂN LOẠI KIỂU

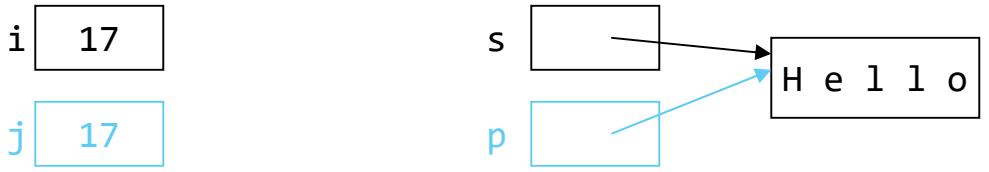



Tất cả các kiểu trên đều tương thích với kiểu *object*

- Có thể được gán với các biến có kiểu *object*
- Các toán tử của kiểu *object* có thể được áp dụng trên chúng

# KIỂU GIÁ TRỊ & KIỂU THAM CHIẾU

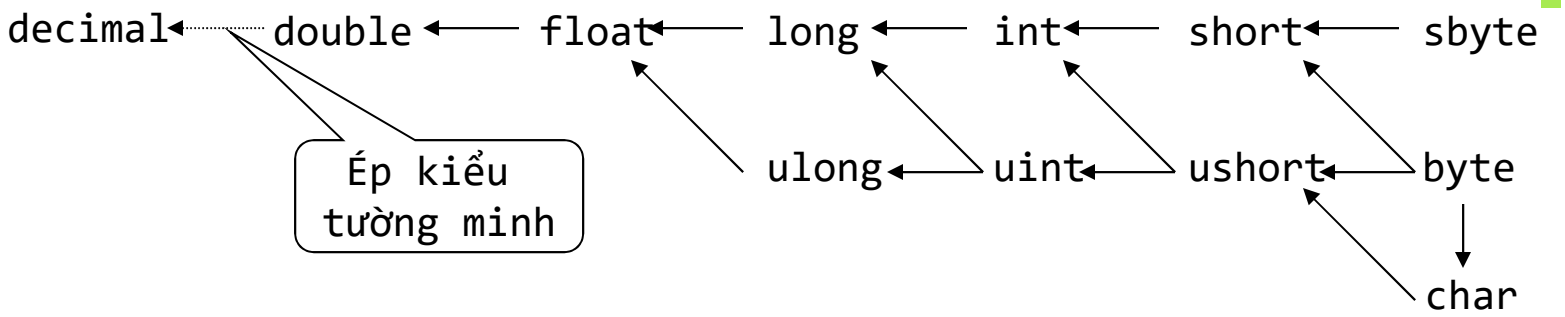
| | Kiểu giá trị | Kiểu tham chiếu |
|---|---|---|
| Biến chứa | Giá trị | Địa chỉ tham chiếu |
| Nơi lưu trữ | Stack | Heap |
| Giá trị khởi tạo | 0, false, '\0' | null |
| Lệnh gán | Sao chép giá trị | Sao chép tham chiếu |
| Ví dụ | int i = 17;<br>int j = i; | string s = "Hello"<br>string p = s; |



# CÁC KIỂU DỰNG SẴN

| Kiểu | Kế thừa từ | Kích thước | Mô tả - giá trị |
|---|---|---|---|
| byte | System.Byte | 1 | Không dấu (0..255) |
| char | System.Char | 1 | Mã ký thự Unicode |
| bool | System.Boolean | 1 | true hay false |
| sbyte | System.SByte | 1 | Có dấu (-128 .. 127) |
| short | System.Int16 | 1 | Có dấu (-32768 .. 32767) |
| ushort | System.UInt16 | 2 | Không dấu (0 .. 65535) |
| int | System.Int32 | 4 | Có dấu (-2147483647 .. 2147483647) |
| uint | System.UInt32 | 4 | Không dấu (0 .. 4294967295) |
| float | System.Single | 4 | Số thực (≈ ±1.5*10-45 .. ≈ ±3.4*1038) |
| double | System.Double | 8 | Số thực (≈ ±5.0*10-324 .. ≈ ±1.7*10308) |
| decimal | System.Decimal | 8 | số có dấu chấm tĩnh với 28 ký số và dấu chấm |
| long | System.Int64 | 8 | Số nguyên có dấu ($-2^{63}$ .. $2^{63}-1$) |
| ulong | System.UInt64 | 8 | Số nguyên không dấu (0 .. $2^{64}-1$) |

# CHUYỂN ĐỔI KIỂU




- Một đối tượng có thể chuyển đổi từ kiểu này sang kiểu khác theo hai cách:
  - Chuyển đổi ngầm định: được thực hiện tự động bởi trình biên dịch.
  - Chuyển đổi tường minh: có sự can thiệp của người lập trình.

```
//Ép kiểu ngầm định
short x = 10;
int y = x;
```

```
//Lỗi ép kiểu
int y = 10;
short x = y;
```

```
//Ép kiểu tường minh
int y = 10;
short x = (int) y;
```

# KIỂU LIỆT KÊ : enum

- Được sử dụng để liệt kê các hằng

Khai báo

```
enum Color {Red, Blue, Green}    // giá trị mặc định: 0, 1, 2
enum Ngay {Hai=2, Ba, Tu, Nam, Sau, Bay, Chunhat=10}
enum Thang:byte {Mot, Hai=2, Ba, Tu, Nam, Sau, Bay=10}
```

Sử dụng

```
Color c = Color.Blue;
int x = (int) Ngay.Hai;
int y = (int) Ngay.Sau
byte z = (byte) Thang.Ba;
```

# TOÁN TỬ KIỂU ENUM

| Toán tử | Ví dụ |
|---|---|
| So sánh | if (c == Color.Red) ...<br>if (c > Color.Red && c <= Color.Green) ... |
| +, - | c = c + 2; |
| ++, -- | c++; c-- |
| & | if ((c & Color.Red) == 0) ... |
| \| | c = c \| Color.Blue; |
| ~ | c = ~ Color.Red; |

- Chương trình không kiểm tra kết quả có phải là một giá trị hợp lệ của kiểu enum không
- Chú ý:
  - enum không được gán cho kiểu int, trừ khi phải ép kiểu tường minh
  - enum kế thừa kiểu object (Equals, ToString, …)
  - Lớp System.enum cung cấp các phương thức : GetName, Format, GetValues,…

# MỘT SỐ KIỂU MỞ RỘNG

- Kiểu xâu (string) : được sử dụng như một kiểu chuẩn string.
  - Khai báo : string s = “Hello” ;
  - Kiểu string không thể chỉnh sửa được (Nếu muốn chỉnh sửa dùng StringBuilder)
  - Toán tử nối chuỗi là phép cộng “+”
  - string là kiểu tham chiếu, nhưng có thể thực hiện phép so sánh
  - Lớp string có nhiều phương thức hữu dụng: Length, CompareTo, IndexOf, StartsWith, Substring…

# MỘT SỐ KIỂU MỞ RỘNG

- Mảng : tập hợp các phần tử có cùng kiểu
  - Khai báo : int [] a; int [][] b;
  - Mỗi thành phần được xác định thông qua chỉ số
  - Có nhiều kiểu mảng : mảng một chiều, mảng nhiều chiều, mảng ziczắc, …
- Danh sách : mảng có kích thước thay đổi.
  - Khai báo : ArrayList a;
- Cấu trúc (struct)
- Lớp (class)

# BIẾN VÀ HẰNG

- Biến là một vùng nhớ dành riêng cho một kiểu dữ liệu nào đó.
  - Biến có thể được gán giá trị và có thể thay đổi giá trị trong quá trình thực hiện chương trình.
  - Biến phải luôn được gán giá trị trước khi được sử dụng.
  - Cú pháp khai báo biến
    - <tên kiểu> <tên biến>
    - Ví dụ : int x,y;
- Hằng là một biến nhưng giá trị của hằng không thể thay đổi trong quá trình thực hiện chương trình.
  - Cú pháp khai báo hằng
    - <const> <tên kiểu> <tên hằng> = <giá trị>
    - Ví dụ : const float so_pi = 3.1415

# BIẾN VÀ HẰNG

- Một số lưu ý khi đặt tên biến và hằng
  - Có ý nghĩa gần với giá trị lưu trữ
  - Tên biến có phân biệt chữ HOA, chữ thường
  - Tuần theo qui tắc đặt tên biến
    - Không có khoảng trắng
    - Không có dấu
    - Không đặt trùng từ khoá
    - Không sử dụng ký tự đặc biệt (#, $, %, +, -, …)
    - Không bắt đầu bằng số

# BIẾN VÀ HẰNG

- Kiểu dữ liệu khai báo biến trong C# đều là kiểu đối tượng
- Mỗi kiểu đối tượng đều đi kèm:
  - Thuộc tính (property)
  - Phương thức (method)

```
int So_a = int.Parse(txtSo_a.Text);
MessageBox.Show("Số a: " + So_a.ToString());
```

# TOÁN TỬ

- Toán tử được ký hiệu bằng một biểu tượng, dùng để thực hiện một hành động
- Toán tử gán: được ký hiệu là dấu "=", là toán tử hai ngôi làm toán hạng bên trái có giá trị bằng toán hạng bên phải.
  - Ví dụ : x = y;
- Toán tử toán học : Là các toán tử thao tác số học.
  - Có 5 toán tử toán học : +,-,*,/ và %
- Toán tử tăng, giảm: Thực hiện các thao tác số học và gán lại giá trị cho chính biến được tính toán
  - Có 5 toán tử tăng, giảm tương ứng với 5 toán tử số học: +=, -=, *=, /=, %=
  - Ví dụ : x = x + 3 tương đương với x += 3;

# TOÁN TỬ

- Toán tử tăng, giảm 1: ++, --
  - Ví dụ: x = x + 1 tương đương với x+=1 hoặc x++
  - Có hai loại tăng, giảm 1 :
    - Tăng, giảm tiền tố : ++x hoặc –-x
    - Tăng, giảm hậu tố : x++ hoặc x—
- Toán tử quan hệ : Dùng để so sánh giữa hai giá trị và trả về một giá trị logic kiểu bool (true hoặc false).
  - == : so sánh bằng
  - != : so sánh khác
  - > : so sánh lớn hơn
  - >= : so sánh lớn hơn hoặc bằng
  - < : so sánh nhỏ hơn
  - <= : so sánh nhỏ hơn hoặc bằng

# TOÁN TỬ

- Toán tử logic : Dùng để kết hợp hai hoặc nhiều biểu thức logic
  - && : kết hợp theo phép and
  - || : kết hợp theo phép or
  - ! : kết hợp theo phép not.
- Toán tử thao tác bit : Dùng để thực hiện các phép toán nhị phân
  - & : phép and bit
  - | : phép or bit
  - ~ : phép not bit
  - ^ : phép xor bit
- Toán tử ba ngôi :
  - Cú pháp : <biểu thức điều kiện>?<biểu thức 1>:<biểu thức 2>
  - Nếu biểu thức điều kiện đúng sẽ trả về giá trị của biểu thức 1, ngược lại trả về giá trị của biểu thức 2.

# ĐỘ ƯU TIÊN TOÁN TỬ

| STT | Loại toán tử | Toán tử | Thứ tự |
|---|---|---|---|
| 1 | Phép toán cơ bản | (x) x.y f(x) a[x] x++ x-- | Trái sang phải |
| 2 | Toán tử một ngôi | + - ! ~ ++x --x (T)x | Trái sang phải |
| 3 | Phép nhân | * / % | Trái sang phải |
| 4 | Phép cộng | + - | Trái sang phải |
| 5 | Dịch bit | >> << | Trái sang phải |
| 6 | Quan hệ | > < >= <= is | Trái sang phải |
| 7 | So sánh bằng | == != | Phải sang trái |
| 8 | Phép toán AND bit | & | Trái sang phải |
| 9 | Phép toán XOR bit | ^ | Trái sang phải |
| 10 | Phép toán OR bit | \| | Trái sang phải |
| 11 | Phép toán logic AND | && | Trái sang phải |
| 12 | Phép toán logic OR | | Trái sang phải |
| 13 | Điều kiện | ? : | Phải sang trái |
| 14 | Phép gán | = += -= *= /= <<= >>= &= ^= \|= | Phải sang trái |

# BÀI TẬP THỰC HÀNH

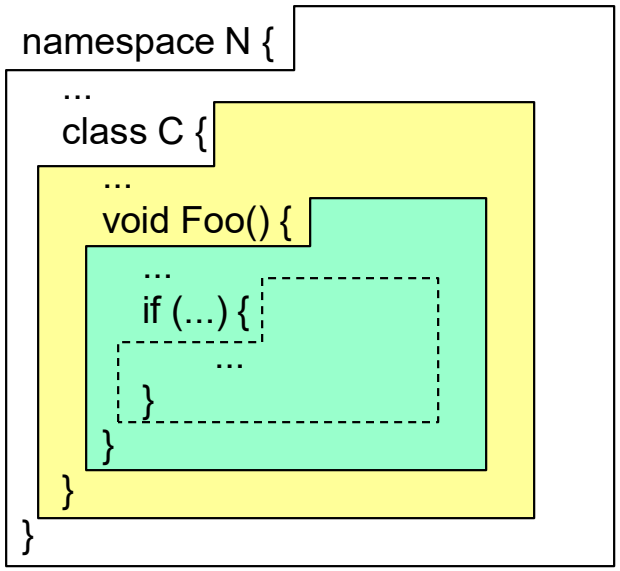
- **Bài 2.2.1**: Viết chương trình nhập vào hai số nguyên a, b và in ra màn hình tổng, hiệu, tích, thương của a và b (Lấy 2 chữ số phần thập phân).

| Đầu vào | Đầu ra |
|---|---|
| Nhập a: 9<br>Nhập b: 5 | 9 + 5 = 14<br>9 – 5 = 4<br>9 * 5 = 45<br>9 / 5 = 1,80 |

# KHAI BÁO BIẾN

# PHẠM VI KHÁI BÁO

- Phân loại phạm vi khai báo
  - namespace: khai báo class, interface, struct, enum, delegate
  - class, interface, struct: khai báo trường dữ liệu, phương thức, ...
  - enumeration: khai báo các hằng thuộc enum
  - khối lệnh: khai báo các biến cục bộ



# CÁC NGUYÊN TẮC KHAI BÁO

- **Nguyên tắc về phạm vi khai báo**
  - Không có hai tên trùng nhau trong cùng một phạm vi và cùng một mức
  - Có thể khai báo trùng tên khi ở các mức khác nhau
- **Nguyên tắc về tầm hoạt động**
  - Tên chỉ có tác dụng trong phạm vi khai báo của nó.
  - Tầm hoạt động còn bị ảnh hưởng bởi các modifier như: *public*, *private*, *protected* và *internal*.

# PHẠM VI KHAI BÁO

- Các namespace ở file khác nhau tạo lên phạm vi khai báo khác nhau
- Namespace lồng tạo nên phạm vi khai báo của riêng nó

File: XXX.cs

```
namespace A {
    ... classes ...
    ... interfaces ...
    ... structs ...
    ... enumerations ...
    ... delegates ...

}
```

File: YYY.cs

```
namespace A {
    ...

}
namespace C {...}
```

# SỬ DỤNG NAMESPACE

*Color.cs*

```
namespace Util {
   public enum Color {...}
}
```

*Figures.cs*

```
namespace Util.Figures {
   public class Rect {...}
   public class Circle
{...}
}
```

*Triangle.cs*

```
namespace Util.Figures {
   public class Triangle
{...}
}
```

```
using Util.Figures;

class Test {
   Rect r;       // không cần sử dụng tên đầy đủ (vì đã sử dụng Util.Figures)
   Triangle t;
   Util.Color c; // tên đầy đủ
}
```

- Đối với các namespace ngoài
  - Phải bổ sung thêm, dùng từ khoá using (e.g. *using Util;)*
  - sử dụng tên đầy đủ (e.g. *Util.Color*)
- Hầu hết các chương trình phải sử dụng namespace System => using System;

# CLASS, INTERFACE, STRUCT

```
class C
{  // có thể áp dụng cho cả struct
    ... Trường dữ liệu, hằng ...
    ... Phương thức ...
    ... Hàm tạo, hàm huỷ ...
    ... Thuộc tính ...
    ... Indexer ...
    ... Sự kiện ...
    ... Chồng toán tử ...
    ... Các kiểu lồng (class, interfaces struct,
    enum, delegate) ...
}
```

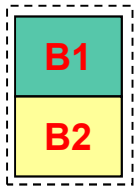
```
interface IX
{
    ... Phương thức
...
    ... Thuộc tính
...
    ... Indexer ...
    ... Sự kiện ...
}
```

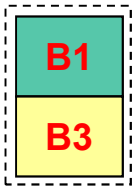
- Phạm vi khai báo của subclass không phụ thuộc vào phạm vi khai báo của base class => có thể khai báo tên trùng nhau trong subclass và base class

# KHỐI LỆNH

Các loại khối lệnh

```
void foo (int x) {                    // khối lệnh của phương thức B1
    ... local variables ...
    if (...) {                        // khối lệnh lồng
        ... local variables ...   B2
    }
    for (int i = 0; ...) {            // khối lệnh lồng
        ... local variables ...   B3
    }
}
```




- Phạm vi khai báo của khối lệnh bao gồm phạm vi khai báo của các khối lệnh lồng trong nó.
- Các tham số hình thức phụ thuộc vào phạm vi khai báo của khối lệnh của phương thức.
- Các biến được sử dụng trong vòng lặp phụ thuộc vào khối lệnh của vòng lặp
- Phải khai báo biến cục bộ trước khi sử dụng chúng

# KHAI BÁO BIẾN CỤC BỘ

```
void foo(int a) {
     int b;
     if (...) { int b; // lỗi: b đã được khai báo ở khối lệnh bên ngoài
          int c; int d; ...
     }
     else {
          int a; // lỗi: a là tham số
          int d; // đúng: không xung đột với d ở lệnh if
     }
     for (int i = 0; ...) {...}
     for (int i = 0; ...) {...} //đúng:không xung đột với i ở vòng lặp trước
     int c;
}
```

# BÀI TẬP THỰC HÀNH

- **Bài 2-2.2**: Nhập từ bàn phím số nguyên x. Tính và in ra màn hình giá trị biểu thức:
  - $A = (x + 3).(x^2 - 3x + 9)$
  - $B = (x^2 + y^2).(x - y) - (x^3 - y^3)$
  - $C = x^2 - 2\sqrt[3]{x}$

# BÀI TẬP THỰC HÀNH

- **Bài 2.2.3**: Viết chương trình nhập 2 phân số từ bàn phím. Tính tổng 2 phân số vừa nhập và hiển thị ra màn hình theo dạng ts/ms (VD: 1/2).

| Đầu vào | Đầu ra |
|---|---|
| Nhập tử số 1: 8<br>Nhập mẫu số 1: 12<br>Nhập tử số 2: 2<br>Nhập mẫu số 2: 5 | Phân số 1: 8/12<br>Phân số 2: 2/5<br>8/12 + 2/5 = 16/60 = 4/15 |

# BÀI TẬP THỰC HÀNH

- **Bài 2.2.4**: Viết chương trình tính chu vi và diện tích Hình chữ nhật với chiều dài và chiều rộng được nhập từ bàn phím. In kết quả ra màn hình với 2 chữ số sau dấu phẩy.

| Đầu vào | Đầu ra |
|---|---|
| Nhập tử chiều dài: 3<br>Nhập chiều rộng: 4 | Chu vi HCN: 14<br>Diện tích HCN: 12 |