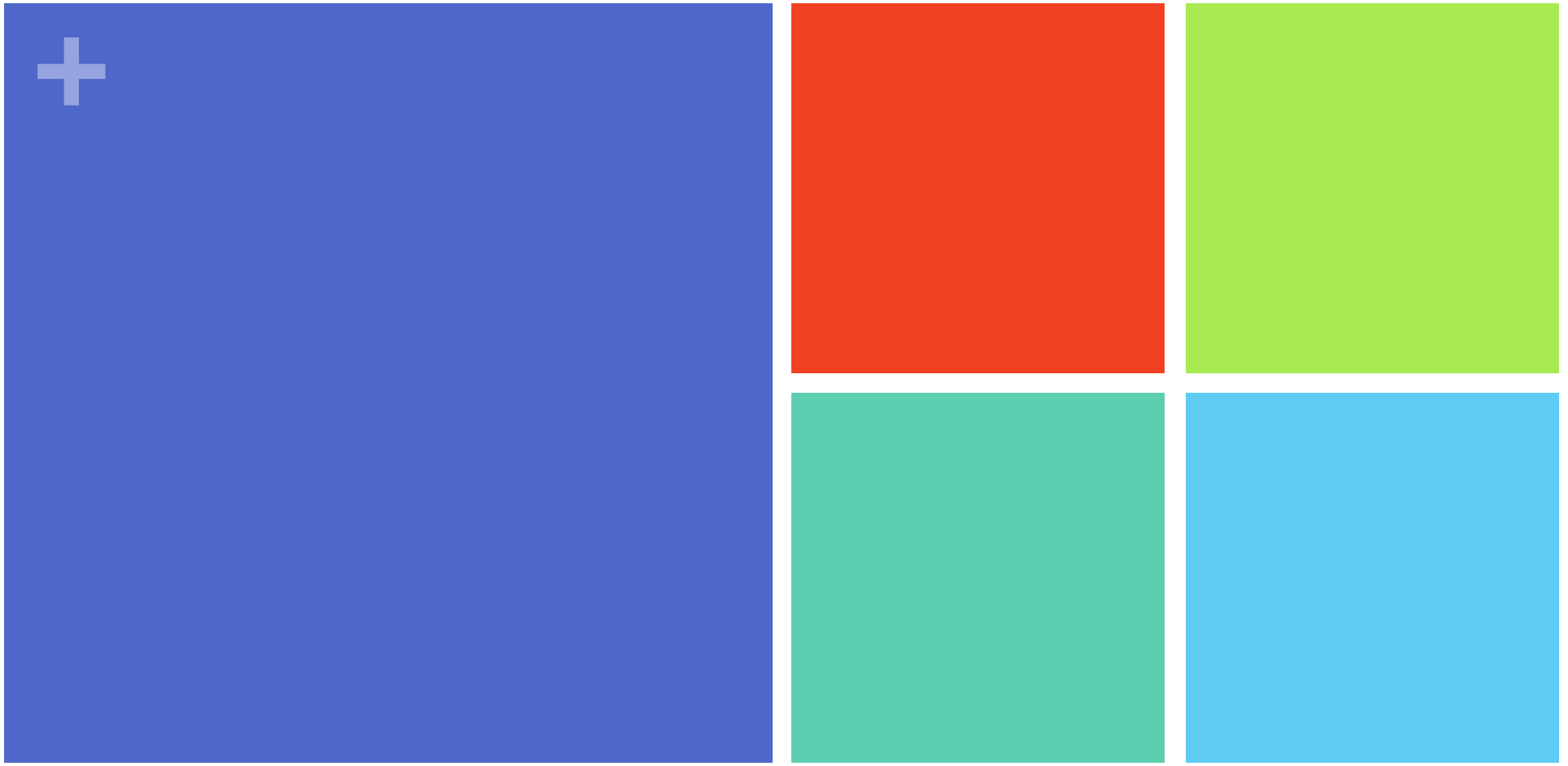
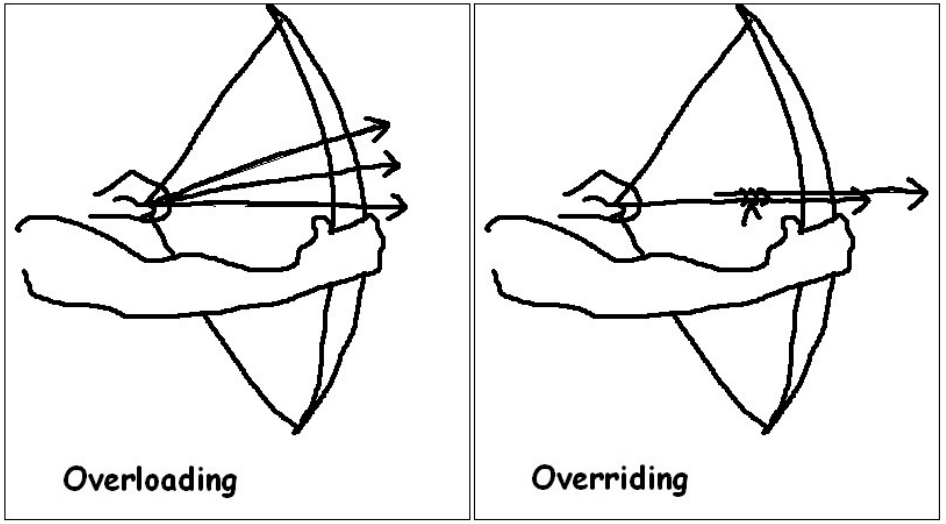


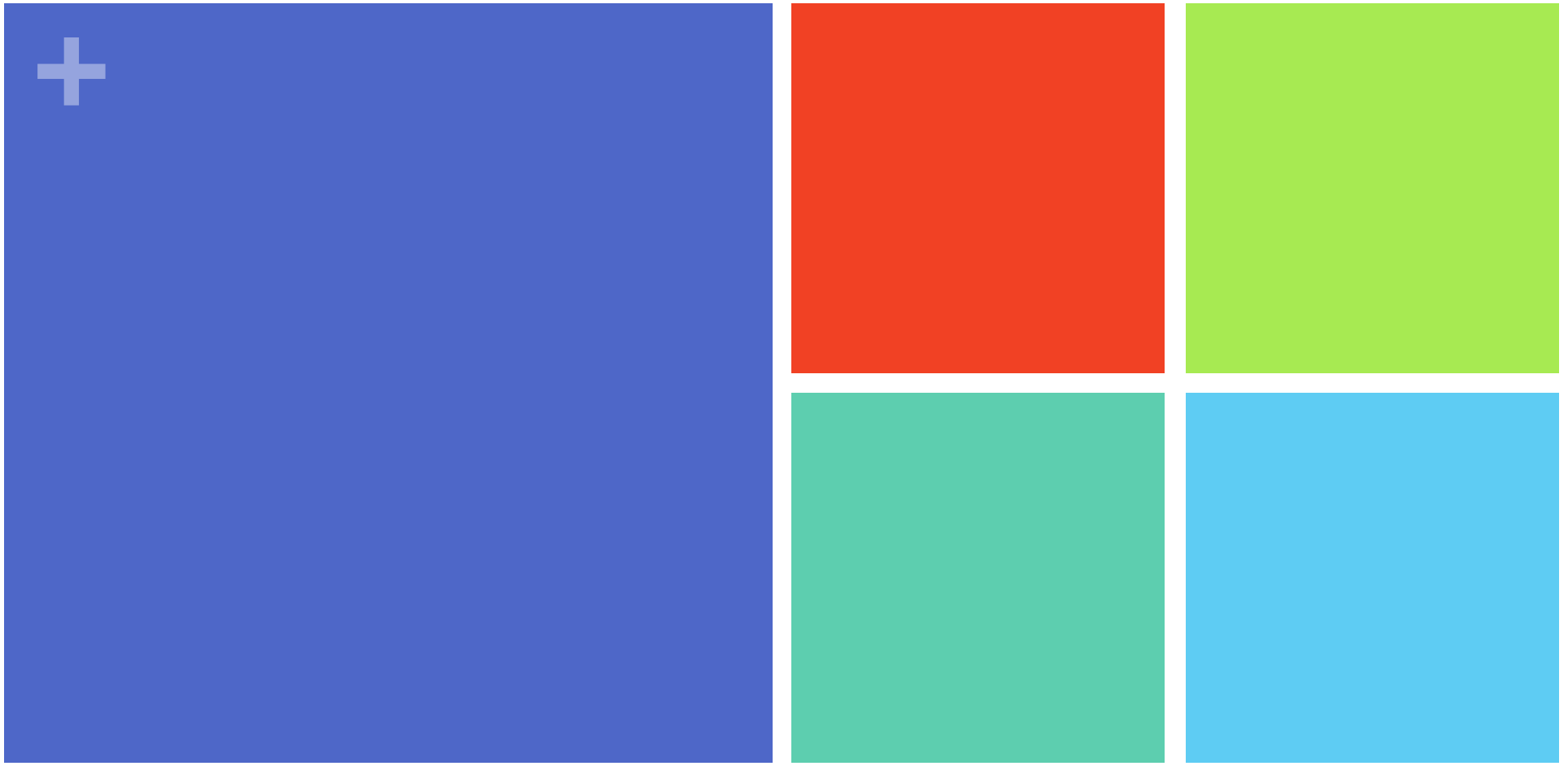
NẠP CHỒNG VÀ KẾ THỪA

TS. ĐẶNG THÀNH TRUNG

NỘI DUNG

- Nạp chồng toán tử
- Kế thừa





NẠP CHỒNG

VÍ DỤ

- Xây dựng lớp phân số có thể thực hiện các phép toán số học : cộng, trừ, nhân chia, …
- Có hai cách xây dựng các phép toán
  - Dùng phương thức :

```
Faction p1, p2, p3
p3 = p1.add(p2);
```

  - Dùng toán tử

```
Faction p1, p2, p3
p3 = p1 + p2;
```

SỬ DỤNG TỪ KHÓA operator

- Các toán tử là các phương thức tĩnh. Giá trị trả về thể hiện kết quả của một toán tử và các tham số là toán hạng.
- Tạo một toán tử cho một lớp tức là thực hiện việc lạp chồng.
- Toán tử nạp chồng được khai báo sử dụng từ khóa **operator**
  - Ví dụ

```
class Fraction {
      int x, y;
      public Fraction (int x, int y) {this.x = x; this.y = y; }
      public static Fraction operator + (Fraction a, Fraction b) {
                  return new Fraction(a.x * b.y + b.x * a.y, a.y * b.y);
      }
}
```

  - Sử dụng

```
Fraction a = new Fraction(1, 2);
Fraction b = new Fraction(3, 4);
Fraction c = a + b; // c.x == 10, c.y == 8
```

MỘT SỐ LUẬT

- Định nghĩa những toán tử trong kiểu dữ liệu giá trị, kiểu do ngôn ngữ xây dựng sẵn
- Cung cấp những phương thức nạp chồng toán tử chỉ bên trong của lớp nơi mà những phương thức được định nghĩa.
- Sử dụng tên và những ký hiệu qui ước mô tả trong Common Language Specification (CLS)
- Phải cung cấp các phương thức thay thế cho toán tử nạp chồng

MỘT SỐ TOÁN TỬ

Biểu tượng	Tên phương thức thay thế	Tên toán tử
+	Add	Toán tử cộng
-	Subtract	Toán tử trừ
*	Multiply	Toán tử nhân
/	Divide	Toán tử chia
%	Mod	Toán tử chia lấy dư
^	Xor	Toán tử or loại trừ
&	BitwiseAnd	Toán tử and nhị phân
|	BitwiseOr	Toán tử or nhị phân
&&	And	Toán tử and logic
||	Or	Toán tử or logic
=	Assign	Toán tử gán
<<	LeftShift	Toán tử dịch trái
>>	RightShift	Toán tử dịch phải

MỘT SỐ TOÁN TỬ

Biểu tượng	Tên phương thức thay thế	Tên toán tử
==	Equals	Toán tử so sánh bằng
>	Compare	Toán tử so sánh lớn hơn
<	Compare	Toán tử so sánh nhỏ hơn
!=	Compare	Toán tử so sánh khác
>=	Compare	Toán tử so sánh lớn hơn hoặc bằng
<=	Compare	Toán tử so sánh nhỏ hơn hoặc bằng
*=	Multiply	Toán tử nhân và gán lại
-=	Subtract	Toán tử trừ và gán lại
^=	Xor	Toán tử or loại trừ và gán lại
<<=	LeftShift	Toán tử dịch trái và gán lại
%=	Mod	Toán tử chia dư rồi gán lại

MỘT SỐ TOÁN TỬ

Biểu tượng	Tên phương thức thay thế	Tên toán tử
+=	Add	Toán tử cộng rồi gán lại
&=	BitwiseAnd	Toán tử and rồi gán lại
|=	BitwiseOr	Toán tử or rồi gán lại
/=	Divide	Toán tử chia rồi gán lại
--	Decrement	Toán tử giảm
++	Increment	Toán tử tăng
-	Negate	Toán tử phủ định một ngôi
+	Plus	Toán tử cộng một ngôi
~	OnesComplement	Toán tử bù

TOÁN TỬ SO SÁNH BẰNG

- Nếu nạp chồng toán tử so sánh bằng (==), nên nạp chồng phương thức ảo **Equals** của lớp object và chuyển lại cho toán tử so sánh bằng thực hiện.
- Ví dụ:

```
public overide bool Equals(object o) {
        if ( !(o is Fraction) ) return false;
        return this == (Fraction) o;
}
```

- Nạp chồng toán tử có tính đối xứng, tức là nếu toán tử so sánh bằng được nạp chồng thì cũng phải nạp chồng toán tử không bằng (!=).
- Nếu toán tử (==) và (!=) mà được nạp chồng và phương thức Equals không được nạp chồng thì chương trình sẽ cảnh báo.

VÍ DỤ

```
using System; public class Fraction {
     public Fraction(int numerator,int denominator) {
          Console.WriteLine("In Fraction Constructor( int, int) ");
          this.numerator = numerator; this.denominator = denominator;
     }
     public Fraction(int wholeNumber) {
          Console.WriLine("In Fraction Constructor( int )");
          numerator = wholeNumber; denominator = 1;
     }
     public static implicit operator Fraction( int theInt ) {
          Console.WriteLine(" In implicit conversion to Fraction");
          return new Fraction( theInt );
     }
```

VÍ DỤ

```
public static explicit operator int( Fraction theFraction ) {
     Console.WriteLine("In explicit conversion to int");
     return theFraction.numerator / theFraction.denominator;
}
public static bool operator == ( Fraction lhs, Fraction rhs) {
     Console.WriteLine("In operator ==");
     if ( lhs.numerator == rhs.numerator && lhs.denominator == rhs.denominator
)
          return true; // thực hiện khi hai phân số không bằng nhau
     return false;
}
public static bool operator != ( Fraction lhs, Fraction rhs) {
     Console.WriteLine("In operator !=");
     return !( lhs == rhs );
}
public override bool Equals( object o ) {
     Console.WriteLine("In method Equals");
     if ( !(o is Fraction )) return false;
     return this == ( Fraction ) o;
}
```

VÍ DỤ

```
public static Fraction operator+( Fraction lhs, Fraction rhs ) {
     Console.WriteLine("In operator +");
     if (lhs.denominator == rhs.denominator )
          return new Fraction(lhs.numerator + rhs.numerator, lhs.denominator
          ); //thực hiện khi hai mẫu số không bằng nhau
     int firstProduct = lhs.numerator * rhs.denominator;
     int secondProduct = rhs.numerator * lhs.denominator;
     return new Fraction( firstProduct + secondProduct, lhs.denominator *
rhs.denominator);
}
public override string ToString() {
     string s = numerator.ToString() + "/" + denominator.ToString();
     return s;
} //biến thành viên lưu tử số và mẫu số
private int numerator;
private int denominator;
}
```

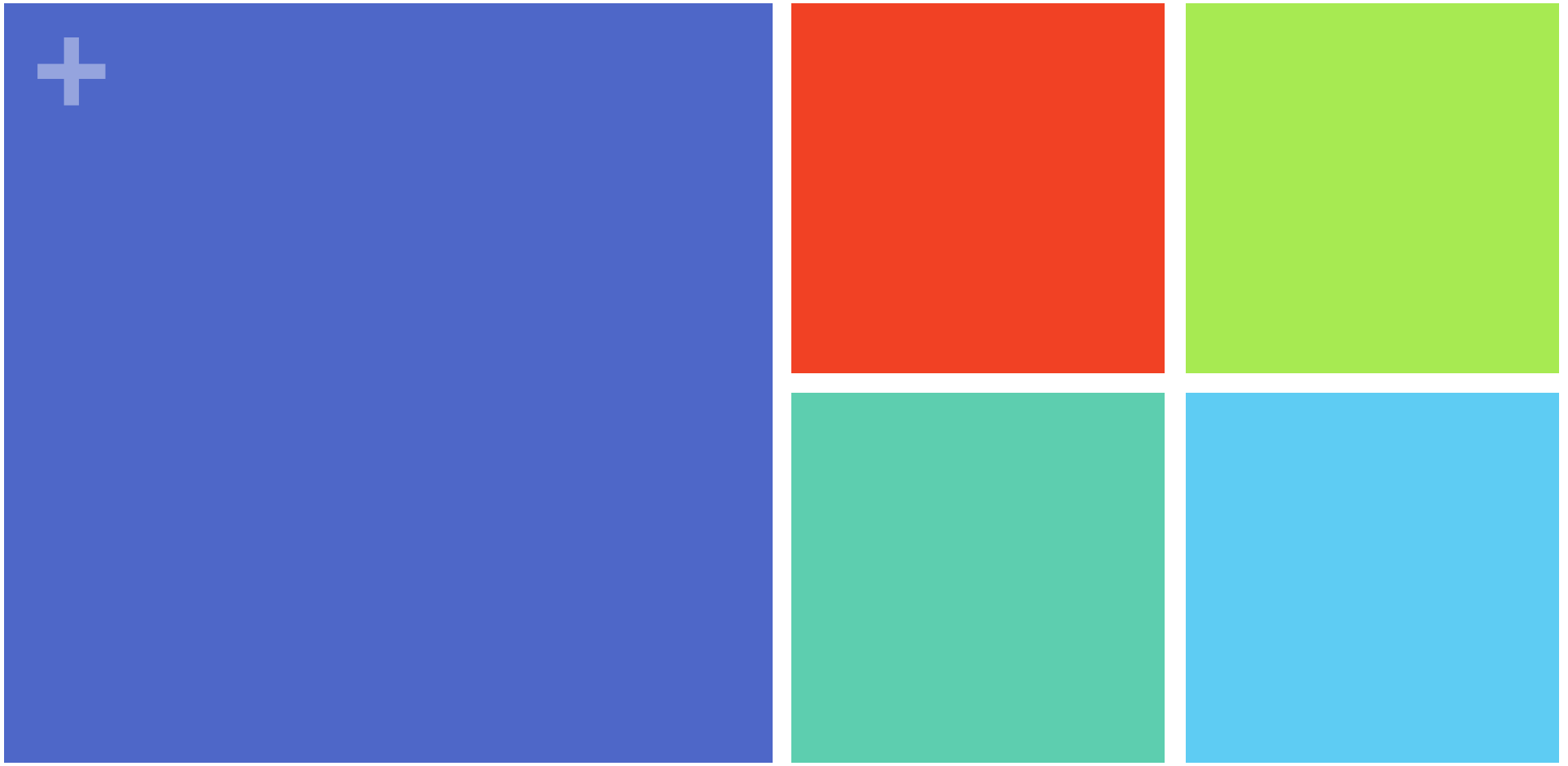
VÍ DỤ

```
public class Tester {
     static void Main() {
          Fraction f1 = new Fraction( 3, 4);
          Console.WriteLine("f1:{0}",f1.ToString());
          Fraction f2 = new Fraction( 2, 4);
          Console.WriteLine("f2:{0}",f2.ToString());
          Fraction f3 = f1 + f2;
          Console.WriteLine("f1 + f2 = f3:{0}",f3.ToString());
          Fraction f4 = f3 + 5;
          Console.WriteLine("f4 = f3 + 5:{0}",f4.ToString());
          Fraction f5 = new Fraction( 2, 4);
          if( f5 == f2 ) {
                    Console.WriteLine("f5:{0}==f2:{1}", f5.ToString(), f2.ToString());
          }
     }
}
```

BÀI TẬP THỰC HÀNH

Bài 2.5.1.1: Áp dụng Nạp chồng vào lớp Fraction để xây dựng các chức năng sau:

- Thuộc tính: tử số, mẫu số (nguyên).
- Phương thức: Nhập dữ liệu từ bàn phím, hiển thị dữ liệu ra màn hình.
- Có phương thức rút gọn phân số.
- **Có phương thức +, -, *, / hai phân số**
- **Có phương thức so sánh hai phân số (>, ≥, <, ≤, ==)**

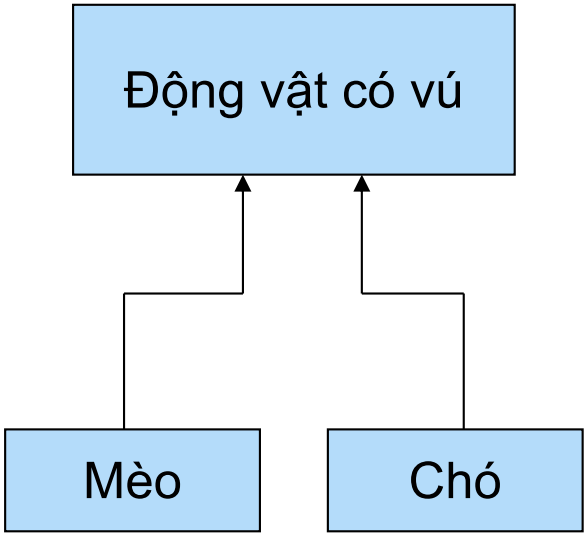


KẾ THỪA

KHÁI NIỆM

- Tính thừa kế là một khái niệm nền tảng cho phép tái sử dụng mã lệnh đang tồn tại và điều này giúp tiết kiệm được thời gian trong việc lập trình

- Các class có thể thừa kế từ class khác. Class mới được gọi là class được dẫn xuất (hay còn gọi là class con) sẽ được quyền truy xuất đến tất cả các thành viên dữ liệu và các phương thức không được biểu thị private của class cơ sở (hay còn gọi là class cha)



KẾ THỪA

- Quan hệ đặc biệt hóa được thực thi bằng cách sử dụng sự ***kế thừa***.
- Trong ngôn ngữ C#, để tạo một lớp kế thừa từ lớp khác, thêm dấu ":" vào sau tên lớp đó

```
public class Meo : Dongvatcovu
```

  - Lớp kế thừa (Meo) gọi là lớp dẫn xuất
  - Lớp được kế thừa (Dongvatcovu) gọi là lớp cha hoặc lớp cơ sở.
- Lớp dẫn xuất kế thừa tất cả các thành phần của lớp cơ sở.
- Lớp dẫn xuất có thể tạo phương thức mới bằng việc đánh dấu với từ khóa **new**

VÍ DỤ

```
class A {// Lớp cơ sở
      public int a;
      public A() {...}
      public void F() {...}
}

class B : A { // Lớp con (kế
      thừa từ A, mở rộng A)
      int b;
      public B() {...}
      public void G() {...}
}
```

- Lớp B kế thừa thuộc tính a và phương thức F, bổ sung thêm thuộc tính b và phương thức G
  - Hàm tạo không được kế thừa
  - Các phương thức được kế thừa có thể chồng lớp con
- Một lớp chỉ được kế thừa từ lớp khác nhưng có thể cài đặt cho nhiều interface.
- Một lớp chỉ có thể kế thừa từ một lớp khác không phải là struct
- Một lớp không có lớp cơ sở tường minh sẽ kế thừa từ lớp ***object***

GÁN VÀ KIỂM TRA KIỂU

- **Khai báo**

```
class A {...}
class B : A {...}
class C: B {...}
```

- **Lệnh gán**

```
A a = new A(); // kiểu static của biến a là kiểu khi khai báo (A)
               // kiểu dynamic type của biến a là kiểu của đối tượng được khởi tạo (A)
a = new B(); // kiểu dynamic là B
a = new C(); // kiểu dynamic là C
B b = a; // lỗi khi dịch chương trình
```

- **Kiểm tra kiểu khi thực thi (Toán tử is)**

```
a = new C();
if (a is C) ... // true, nếu kiểu dynamic của a là C hoặc subclass của C
if (a is B) ... // true
if (a is A) ... // true
a = null;
if (a is C) ... // false
```

ÉP KIỂU

- Ép kiểu

```
A a = new C();
B b = (B) a; // đúng, nếu a có kiểu dynamic là B hoặc subclass của B
C c = (C) a;
a = null;
c = (C) a; // đúng, c = null
```

- Toán tử as

```
A a = new C();
B b = a as B; // if (a is B) b = (B)a; else b = null;
C c = a as C;
a = null;
c = a as C; // c == null
```

NẠP CHỒNG PHƯƠNG THỨC

Chỉ những phương thức được khai báo với từ khoá virtual mới được chồng.

```
class A { public void F() {...} // F không được chồng
public virtual void G() {...} // Có thể chồng G trong subclass }
```

Các phương thức chồng được khai báo với từ khoá override

```
class B : A {
    public void F() {...} // nên sử dụng từ khoá new
    public void G() {...} // nên sử dụng từ khoá new
    public override void G() { // ok: chồng toán tử G
    ...
    base.G(); // gọi G() của lớp cơ sở }
}
```

Thuộc tính và indexer cũng được chồng.

- Phương thức static không được chồng

HÀM TẠO VÀ THỪA KẾ

Gọi không tường minh tới hàm tạo của baseclass

```
class A {
   ...
}

class B : A {
   public B(int x)
}
```

```
B b = new B(3);
```

Đúng
- A có hàm tạo mặc định A()
- B(int x)

```
class A {
   public A() {...}
}

class B : A {
   public B(int x)
}
```

```
B b = new B(3);
```

Đúng
- A()
- B(int x)

```
class A {
   public A(int x) {...}
}

class B : A {
   public B(int x) {...}
}
```

```
B b = new B(3);
```

Lỗi
- Hàm tạo mặc định của A không tồn tại

Gọi tường minh

```
class A {
   public A(int x) {...}
}

class B : A {
   public B(int x)
   : base(x) {...}
}
```

```
B b = new B(3);
```

Đúng
- A(int x)
- B(int x)

LỚP TRỪU TƯỢNG

- Lớp trừu tượng thực chất là lớp cơ sở (base class) mà các class khác có thể được dẫn xuất từ nó.
- Các lớp không phải là lớp trừu tượng (non-abstract class) được gọi là lớp cụ thể (concrete class)
- Lớp trừu tượng có thể có cả hai loại phương thức: phương thức trừu tượng và phương thức cụ thể.
- Một kiểu được dẫn xuất từ một lớp cơ sở trừu tượng thừa kế tất cả các thành viên kiểu cơ sở bao gồm sự thực thi mọi phương thức

LỚP TRỪU TƯỢNG

- Khi nào thì sử dụng lớp trừu tượng?
  - Nếu muốn tạo các lớp mà các lớp này sẽ chỉ là các lớp cơ sở, và không muốn bất cứ ai tạo các đối tượng của các kiểu lớp này.
  - Lớp trừu tượng thường được dùng để biểu thị rằng nó là lớp không đầy đủ và rằng nó được dự định sẽ chỉ được dùng như là một lớp cơ sở.

LỚP TRỪU TƯỢNG

```
abstract class Stream {
     public abstract void Write(char ch);
     public void WriteString(string s) { foreach (char ch in s) Write(s); }
}
class File : Stream {
     public override void Write(char ch) {... ghi ch vào bộ nhớ ...}
}
```

Chú ý

- Các phương thức abstract không có phần cài đặt.
- Nếu một lớp có phương thức abstract (do khai báo hoặc do kế thừa) thì lớp đó phải là lớp trừu tượng.
- Không thể tạo ra đối tượng của lớp trừu tượng

THUỘC TÍNH VÀ INDEXER TRỪU TƯỢNG

```
abstract class Sequence {
     public abstract void Add(object x);  // phương thức
     public abstract string Name { get; } // thuộc tính
     public abstract object this [int i] { get; set; }       // indexer
}

class List : Sequence {
     public override void Add(object x) {...}
     public override string Name { get {...} }
     public override object this [int i] { get {...} set {...} }
}
```

Chú ý:

- Các thuộc tính và indexer được chồng phải có cùng các phương thức get/set tương tự như trong baseclass

LỚP TRỪU TƯỢNG

Abstract Method	Virtual Method
từ khoá: abstract	từ khoá: virtual
Chỉ có phần khai báo method và kết thúc là dấu ";", Không cần có phần thực thi cho phương thức abstract ở lớp abstract	Có phần thực thi cho phương thức virtual ở lớp cơ sở
Bắt buộc lớp dẫn xuất phải override lại	Không bắt buộc lớp dẫn xuất phải override
từ khoá override trước phương thức ở lớp con	từ khoá override trước phương thức ở lớp con

LỚP CÔ LẬP (SEALED CLASS)

- Từ khóa **sealed** được sử dụng để biểu thị khi khai báo một lớp nhằm ngăn ngừa sự dẫn xuất từ một lớp, điều này cũng giống như việc ngăn cấm một lớp nào đó có lớp con.
- Một lớp cô lập cũng không thể là một lớp trừu tượng.
- Các kiểu struct trong C# được ngầm định là **sealed**. Do vậy, chúng không thể được thừa kế.

LỚP CÔ LẬP (SEALED CLASS)

- Ví dụ

```
sealed class Account : Asset {
    long val;
    public void Deposit (long x) { ... }
    public void Withdraw (long x) { ... }
    ...
}
```

- Chú ý
  - Không thể kế thừa từ *sealed* class, nhưng sealed class vẫn kế thừa từ lớp khác.
  - Cũng có thể sử dụng từ khoá sealed với các phương thức

LỚP TỔNG QUAN (System.object)

- System.Object là baseclass của tất cả các lớp

```
class Object {
     protected object   MemberwiseClone() {...}
     public Type        GetType() {...}
     public virtual bool          Equals (object o) {...}
     public virtual string        ToString() {...}
     public virtual int           GetHashCode() {...}
}
```

Sử dụng trực tiếp

```
Type t = x.GetType();  trả về kiểu được khai báo của x
object copy = x.MemberwiseClone();        tạo ra bản sao
```

Chồng trong subclass:

```
x.Equals(y)              so sánh giá trị của x và y
x.ToString()             trả về xâu biểu diễn x
int code = x.getHashCode();      trả về mã băm của x
```

VÍ DỤ VỀ LỚP System.object

```
class Fraction {
      int x, y;
      public Fraction(int x, int y) {
            this.x = x; this.y = y;
      }
      ...
      public override string ToString() {
            return String.Format("{0}/{1}", x,
      y);
      }
      public override bool Equals(object o) {
            Fraction f = (Fraction)o;
            return f.x == x && f.y == y;
      }
      public override int GetHashCode() {
            return x ^ y;
      }
      public Fraction ShallowCopy() {
            return (Fraction) MemberwiseClone();
      }
}
```

```
class Client {
     static void Main() {
          Fraction a = new Fraction(1, 2);
          Fraction b = new Fraction(1, 2);
          Fraction c = new Fraction(3, 4);
          Console.WriteLine(a.ToString());
          Console.WriteLine(a); // (ToString() được gọi
    tự động)
          Console.WriteLine(a.Equals(b)); // true
          Console.WriteLine(a == c); // false
          Console.WriteLine(a.GetHashCode()); // 3
          a = c.ShallowCopy();
          Console.WriteLine(a); // 3/4
     }
}
```

BÀI TẬP THỰC HÀNH

Bài 2.5.1.2: Xây dựng lớp ngày tháng (đặt tên là **TDate**) có các thành phần:

- Thuộc tính: ngày(int), tháng(int), năm(int)
- Có hàm tạo có tham số mặc định và hàm tạo sao chép
- Có các phương thức: Nhập dữ liệu từ bàn phím, hiển thị dữ liệu ra màn hình dưới dạng dd/mm/yyyy.
- Có phương thức so sánh hai ngày tháng
- Có phương thức tính khoảng cách giữa hai ngày tháng (theo ngày)

BÀI TẬP THỰC HÀNH

Bài 2.5.1.2: Trong hàm main(), khai báo hai đối tượng ngày tháng d1, d2 và thực hiện các công việc sau:

- Sử dụng hàm nhập và hiển thị dữ liệu để nhập dữ liệu và hiển thị thông tin của các đối tượng ra màn hình.
- Giữa hai ngày tháng vừa nhập vào, chỉ ra ngày tháng nào sớm hơn.
- Tính khoảng cách giữa hai ngày tháng vừa nhập.

BÀI TẬP THỰC HÀNH

Bài 2.5.1.3: Tạo một lớp có tên là **Person** để mô tả các thông tin về một người bao gồm các thành phần:

- Thuộc tính: Họ tên(kiểu string), Ngày sinh(kiểu **TDate**), Quê quán(kiểu string).
- Có hàm tạo có tham số, hàm tạo sao chép.
- Có phương thức get, set dữ liệu đối tượng.

BÀI TẬP THỰC HÀNH

Bài 2.5.1.3: Kế thừa lớp **Person** để xây dựng lớp **Student**.

Bổ sung thêm thông tin:

- Thuộc tính: điểm trung bình
- Hai hàm thành phần:
  - (1) nhập dữ liệu
  - (2) kiểm tra sinh viên này có phải là kiệt xuất không (là kiệt xuất nếu điểm trung bình lớn hơn 9.5)

BÀI TẬP THỰC HÀNH

Bài 2.5.1.3: Kế thừa lớp **Person** để xây dựng lớp **Professor**. Bổ sung thêm thông tin:

- Thuộc tính: số lượng bài báo đã xuất bản
- Hai hàm thành phần:
  - (1) nhập dữ liệu
  - (2) kiểm tra giáo sư này có phải là kiệt xuất không (là kiệt xuất nếu số lượng bài báo >= 100)

BÀI TẬP THỰC HÀNH

Bài 2.5.1.3: Sử dụng các lớp vừa xây dựng

- Nhập vào một danh sách gồm n người (bao gồm cả Sinh viên và Giáo sư, tất nhiên bạnđược biết ai thuộc loại nào).
- Sau đó bạn đưa ra màn hình thông tin về những người kiệt xuất.