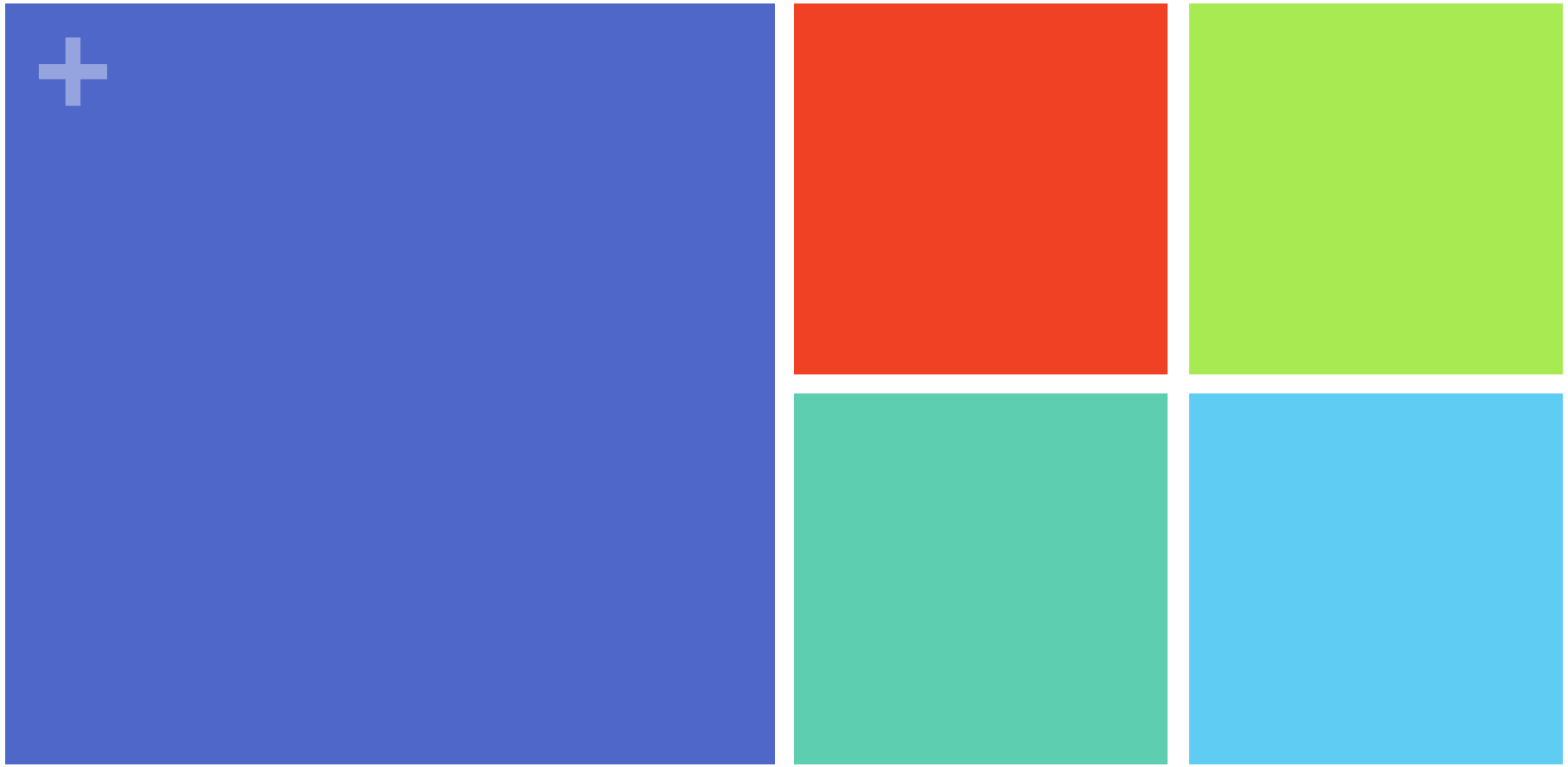
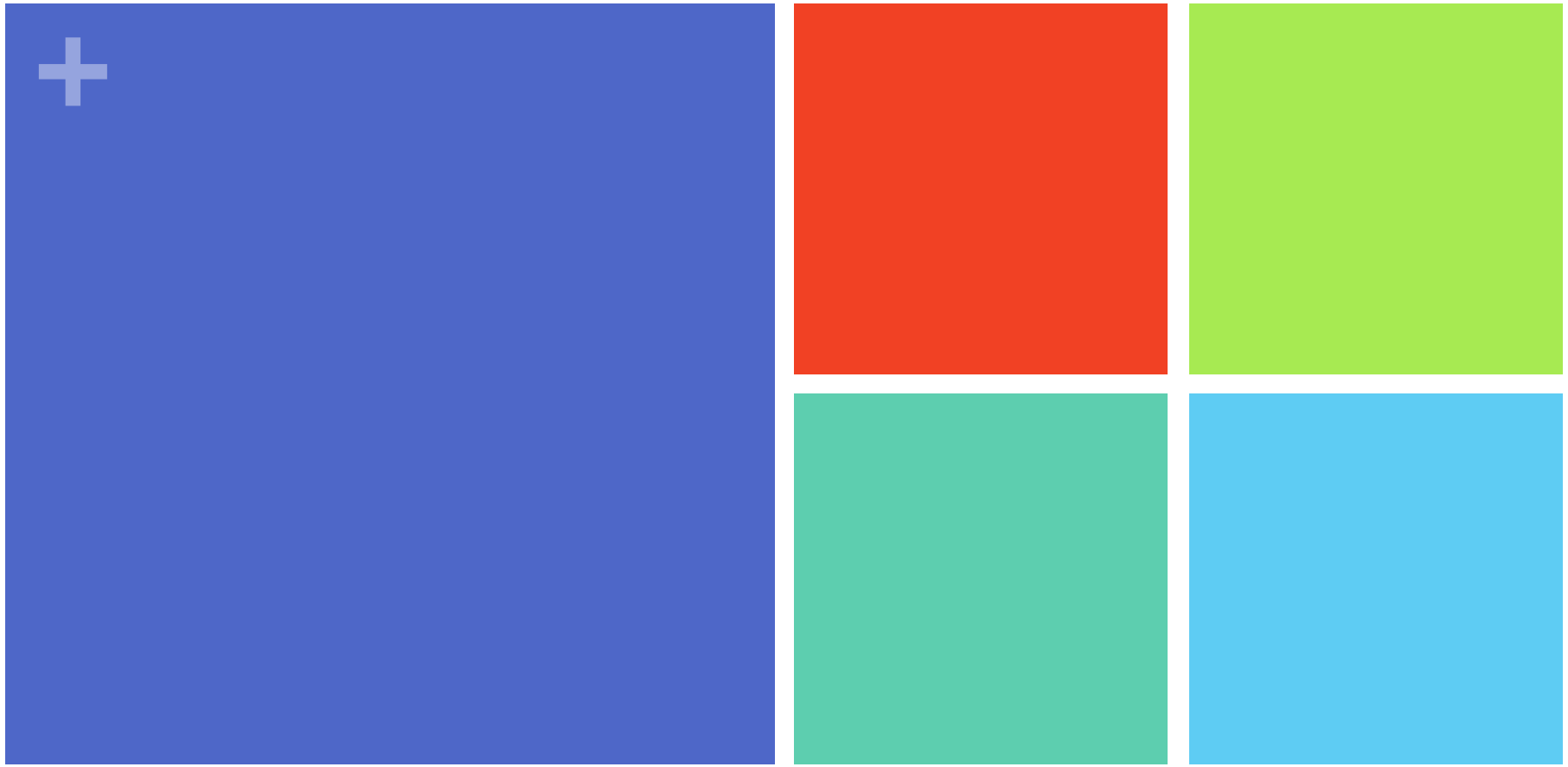


KIỂU DỮ LIỆU CLASS & STRUCT

TS. ĐẶNG THÀNH TRUNG

NỘI DUNG

- Kiểu dữ liệu Lớp
- Kiểu dữ liệu struct
- Boxing và Unboxing



KIỂU DỮ LIỆU LỚP

KIỂU DỮ LIỆU LỚP

- Khai báo lớp

```
[Thuộc tính] [Bổ sung truy cập] class <Định danh lớp> [: Lớp cơ sở]
{
    <Phần thân của lớp: bao gồm định nghĩa các thuộc tính và
    phương thức hành động >
}
```

- Trong C#, phần kết thúc của lớp không có dấu ";" giống như trong C++. Tuy nhiên nếu người lập trình đưa vào thì chương trình dịch vẫn không báo lỗi.
- Trong C#, mọi chuyện đều xảy ra trong một lớp.
- Lớp chỉ kế thừa từ một lớp khác
- Lớp có thể cài đặt cho nhiều interface.
- Một đối tượng gọi là một thể hiện của lớp.
  - Một đối tượng được cấp phát vùng nhớ trên heap
  - Đối tượng phải được tạo với từ khóa new

VÍ DỤ VỀ LỚP

```
class Thoigian {
     int nam, thang, ngay; //các thuộc tính
     int gio, phut, giay;
     public thoiGianHienHanh() { //một phương thức
     System.Console.WriteLine("Hien thi thoi gian !");
     }
}
public class Tester {
     static void Main() {
          Thoigian t = new Thoigian(); t.thoiGianHienHanh();
     }
}
```

THUỘC TÍNH TRUY CẬP

- Thuộc tính truy cập quyết định khả năng truy cập vào lớp từ các yếu tố bên ngoài
- Trong C#,có các thuộc tính truy cập như sau:
  - public : Không hạn chế truy cập
  - private : Chỉ các thành phần bên trong lớp mới được truy cập.
  - protect : Chỉ các thành phần bên trong lớp hoặc lớp kế thừa có thể truy cập.
  - internal : Chỉ các thành phần của lớp cùng khối khai báo có thể truy cập
  - protected internal: chỉ các thành phần của lớp cùng khối khai báo hoặc kế thừa có thể truy cập.
- Khi không khai báo rõ thuộc tính truy cập, các thành phần sẽ được xem như là private.

THUỘC TÍNH

```
class A {
int value = 0;     Trường dữ liệu
          - Khởi tạo giá trị là tuỳ chọn
          - Giá trị khởi tạo phải được tính vào lúc dịch chương trình
          - Không phải khởi tạo giá trị của trường dữ liệu của struct
const long size = ((long)int.MaxValue + 1) / 4;
          Hằng
          - Phải được khởi tạo
          - Giá trị khởi tạo phải được tính vào lúc dịch chương trình
readonly DateTime date;
          Trường dữ liệu Read-only
          - phải được khởi tạo trong khi khai báo hoặc trong hàm tạo
          - Giá trị không cần tính được ngay khi dịch chương trình
          - Giá trị không thể thay đổi
}
```

Truy nhập trong class A

... value ... size ... date ...

Truy nhập từ class khác

A a = new A ();

... a.value ... a.size ... a.date ...

THUỘC TÍNH TĨNH

Phụ thuộc vào class, không phụ thuộc vào đối tượng

```
class Rectangle {
    static Color defaultColor; // mỗi class chỉ có một
    static readonly int scale; // mỗi class chỉ có một
    int x, y, width,height; // mỗi đối tượng có một ...
}
```

Hằng không được khai báo với từ khoá static

Truy nhập trong class

... defaultColor ... scale ...

Truy nhập từ class khác

... Rectangle.defaultColor ... Rectangle.scale .

ĐÓNG GÓI DỮ LIỆU

- Thực hiện truy cập vào các biến thành viên của lớp thông qua các phương thức nhằm bảo mật dữ liệu của lớp
- Có hai bộ truy cập được thiết lập
  - Bộ truy cập lấy dữ liệu
  - Bộ truy cập thiết lập dữ liệu
- Ví dụ

Khai báo

```
public class Time {
     //biến thành viên
     private int hour, minute, second;
     public int Hour {
          get {return hour;}
          set {hour = value;}
     }
}
```

Sử dụng

```
Time t = new Time();
int theHour = t.Hour;
theHour ++;
t.Hour = theHour;
```

ĐÓNG GÓI DỮ LIỆU

- Đóng gói dữ liệu giúp người lập trình kiểm tra trường dữ liệu trước khi sử dụng
- Đóng gói dữ liệu được sử dụng để thiết lập quyền readonly hoặc writeonly cho thuộc tính.

```
public class Time {
    //biến thành viên
    private int hour, minute, second;
    public int Hour {
        get {return hour;}
    }
    public int Minute {
        set {minute = value;}
    }
}
```

Thuộc tính Hour là readonly

Thuộc tính Minute là writeonly

PHƯƠNG THỨC

```
class C {
     int sum = 0, n = 0;

     public void Add (int x) {
          sum = sum + x; n++;
     }

     public float Mean() {
          return (float)sum/n;
     }
}
```

Truy nhập trong class

```
Add(3);
float x = Mean();
```

Truy nhập từ class khác

```
C c = new C();
c.Add(3);
float x = c.Mean();
```

PHƯƠNG THỨC TĨNH

- Chỉ làm việc trên các thành phần dữ liệu static

```
class Rectangle {
    static Color defaultColor;

    public static void ResetColor() {
        defaultColor = Color.white;
    }
}
```

Truy nhập trong class

```
ResetColor();
```

Truy nhập từ class khác

```
Rectangle.ResetColor();
```

THAM SỐ PHƯƠNG THỨC

- Một phương thức có thể có số lượng tham số tùy ý.
- Mỗi tham số phải được khai báo cùng kiểu dữ liệu
- Các tham số được xem như các biến cục bộ
- Có ba kiểu tham số:

Tham trị (tham số vào)

```
void Inc(int x) {x = x + 1;}
void f() {
   int val = 3;
   Inc(val); // val == 3
}
```

- Truyền giá trị
- Tham số hình thức là bản sao của tham số thực
- Tham số thực là một biểu thức

THAM SỐ PHƯƠNG THỨC

Tham chiếu (tham số đệm)

```
void Inc(ref int x) { x = x + 1; }
void f() {
   int val = 3;
   Inc(ref val); // val == 4
}
```

- Truyền tham chiếu
- Tham số hình thức là bí danh của tham số thực. (Truyền địa chỉ của tham số thực)
- Tham số thực phải là biến

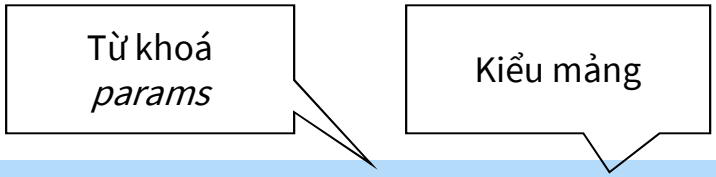
Tham số hiển thị

```
void Read (out int first, out int next)
{
   first = Console.Read();
   next = Console.Read();
}
void f() {
   int first, next;
   Read(out first, out next);
}
```

- Tương tự như tham chiếu nhưng không truyền giá trị khi gọi hàm
- Phải được sử dụng trong phương thức trước khi lấy giá trị.

THAM SỐ PHƯƠNG THỨC

- Tham số có số lượng thay đổi




```
void Add (out int sum, params int[] val) {
        sum = 0;
        foreach (int i in val) sum = sum + i;
}
```

params không được sử dụng với tham chiếu (ref) và tham số hiển thị (out)

Sử dụng

```
Add(out sum, 3, 5, 2, 9);  // sum = 19
```

BÀI TẬP THỰC HÀNH

Bài 2.5.1: Xây dựng lớp **Fraction** mô tả phân số với những thành phần 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 +, -, *, / và so sánh hai phân số.

BÀI TẬP THỰC HÀNH

Bài 2.5.1: Sử dụng lớp **Fraction** và thực hiện các yêu cầu sau:

- Nhập từ bàn phím N phân số (0 < N < 100) và hiển thị N phân số vừa nhập ra màn hình (Tối giản phân số)
- Tìm phân số có giá trị lớn nhất và nhỏ nhất
- Tính tổng, hiệu, tích, thương của 2 phân số max và min
- Sắp xếp N phân số đã nhập theo chiều không tăng.

NẠP CHỒNG PHƯƠNG THỨC

- Nạp chồng phương thức là cách khai báo các phương thức có thể trùng tên nhau.
- Các phương thức có thể khai báo trùng tên nhau nếu
  - Số lượng tham số của chúng khác nhau
  - Kiểu dữ liệu của các tham số là khác nhau
  - Kiểu truyền tham số khác nhau

Khai báo

```
void F (int x) {...}
void F (char x) {...}
void F (int x, long y)
{...}
void F (long x, int y)
{...}
void F (ref int x) {...}
```

Gọi phương thức

```
int i; long n; short s;
F(i);     // F(int x)
F('a');   // F(char x)
F(i, n); // F(int x, long y)
F(n, s); // F(long x, int y);
F(i, s); // nhập nhằng giữa F(int x, long y) và
F(long    x, int y); => lỗi khi dịch
F(i, i); // nhập nhằng giữa F(int x, long y) và
F(long    x, int y); => lỗi khi dịch
```

NẠP CHỒNG PHƯƠNG THỨC

- Các hàm chồng chỉ khác nhau về kiểu dữ liệu trả về không được chấp nhận
  - Khai báo :

```
int F();
string F();
```

  - Gọi hàm

```
F();      //lỗi khi dịch
```

- Các hàm chồng sau cũng không hợp lệ
  - Khai báo

```
void P(int[] a) {...}
void P(params int[] a) {...}
```

  - Gọi hàm

```
int[] a = {1, 2, 3};
P(a);
P(1, 2, 3);
```

HÀM KHỞI TẠO CỦA LỚP

- Phương thức được thực hiện khi người dùng tạo ra một đối tượng gọi là hàm khởi tạo của lớp
- Hàm khởi tạo cũng bị nạp chồng
- Trong hàm khởi tạo có thể gọi hàm khởi tạo khác, sử dụng từ khóa this

```
class Rectangle {
      int x, y, width, height;
      public Rectangle (int x, int y, int w, int h) {
      this.x = x;
      this.y = y;
      width = x;
      height = h;
      }
      public Rectangle (int w, int h) : this(0, 0, w, h) {}
      public Rectangle () : this(0, 0, 0, 0) {}
            ...
}
```

```
Rectangle r1 = new Rectangle();
Rectangle r2 = new Rectangle(2, 5);
Rectangle r3 = new Rectangle(2, 2, 10, 5);
```

HÀM KHỞI TẠO MẶC ĐỊNH

- Nếu người dùng không định nghĩa hàm khởi tạo khi xây dựng lớp thì hệ thống sử dụng hàm khởi tạo mặc định, không có tham số.

```
class C { int x; }
C c = new C();          // ok
```

- Hàm tạo mặc định khởi tạo trường dữ liệu theo nguyên tắc sau:

số	0
enum	0
bool	false
char	'\0'
tham chiếu	null

- Nếu class khai báo hàm tạo, thì hàm tạo mặc định sẽ không được tạo ra

```
class C {
    int x;
    public C(int y) { x = y; }
}
C c1 = new C();         // lỗi khi dịch chương trình
C c2 = new C(3);        // ok
```

HÀM KHỞI TẠO TĨNH

- Không có tham số và không sử dụng từ khoá *public* hoặc *private*
- Mỗi class/struct chỉ có **một** hàm tạo static
- Nó chỉ được viện dẫn một lần trước khi class/struct được sử dụng.
- Được sử dụng để khởi tạo các trường dữ liệu static

```
class Rectangle {
     ...
     static Rectangle() {
     Console.WriteLine("Rectangle
initialized");
     }
}
```

```
struct Point {
     ...
     static Point() {
      Console.WriteLine("Point
initialized");
     }
}
```

HÀM KHỞI TẠO SAO CHÉP

- Hàm khởi tạo sao chép thực hiện sao chép tất cả các biến từ một đối tượng đã có và cùng một kiểu dữ liệu.

```
class Rectangle {
     int x, y, width, height;
     public Rectangle (Rectangle rc) {
     x = rc.x;
     y = rc.y;
     width = rc.width;
     height = rc.height;
     }
     …
}
```

```
Rectangle r1 = new Rectangle();
Rectangle r2 = new Rectangle(r1);
```

TỪ KHÓA this

- Từ khóa this được dùng để tham chiếu đến thể hiện hiện thời của một đối tượng
- Từ khóa this được xem là con trỏ ẩn đến tất cả các phương thức không có thuộc tính tĩnh trong một lớp.
- Tham chiếu this được sử dụng theo ba cách
  - Sử dụng khi các biến thành viên bị che lấp bởi tham số đưa vào
  - Sử dụng tham chiếu this để truyền đối tượng hiện hành vào một tham số của một phương thức đối tượng khác
  - Sử dụng tham chiếu this như là mảng chỉ mục (indexer)

```
class A {
     int x;
     public void setVal(int x) {
          this.x = x;
     }
}
```

```
class B {public void method(A a) {…}}
class A {
     public void setVal(B b) {
          b.method(this);
     }
}
```

HÀM HỦY CỦA LỚP

- Được gọi trên một đối tượng trước khi bị xoá bởi garbage collector.
- Hàm huỷ của lớp cơ sở được tự động gọi vào cuối cùng
- Không sử dụng từ khoá *public* hoặc *private*.
- Struct không cần có hàm huỷ

```
class Test {
    ~Test() {
    ... cleanup actions ...
    }
}
```

BÀI TẬP THỰC HÀNH

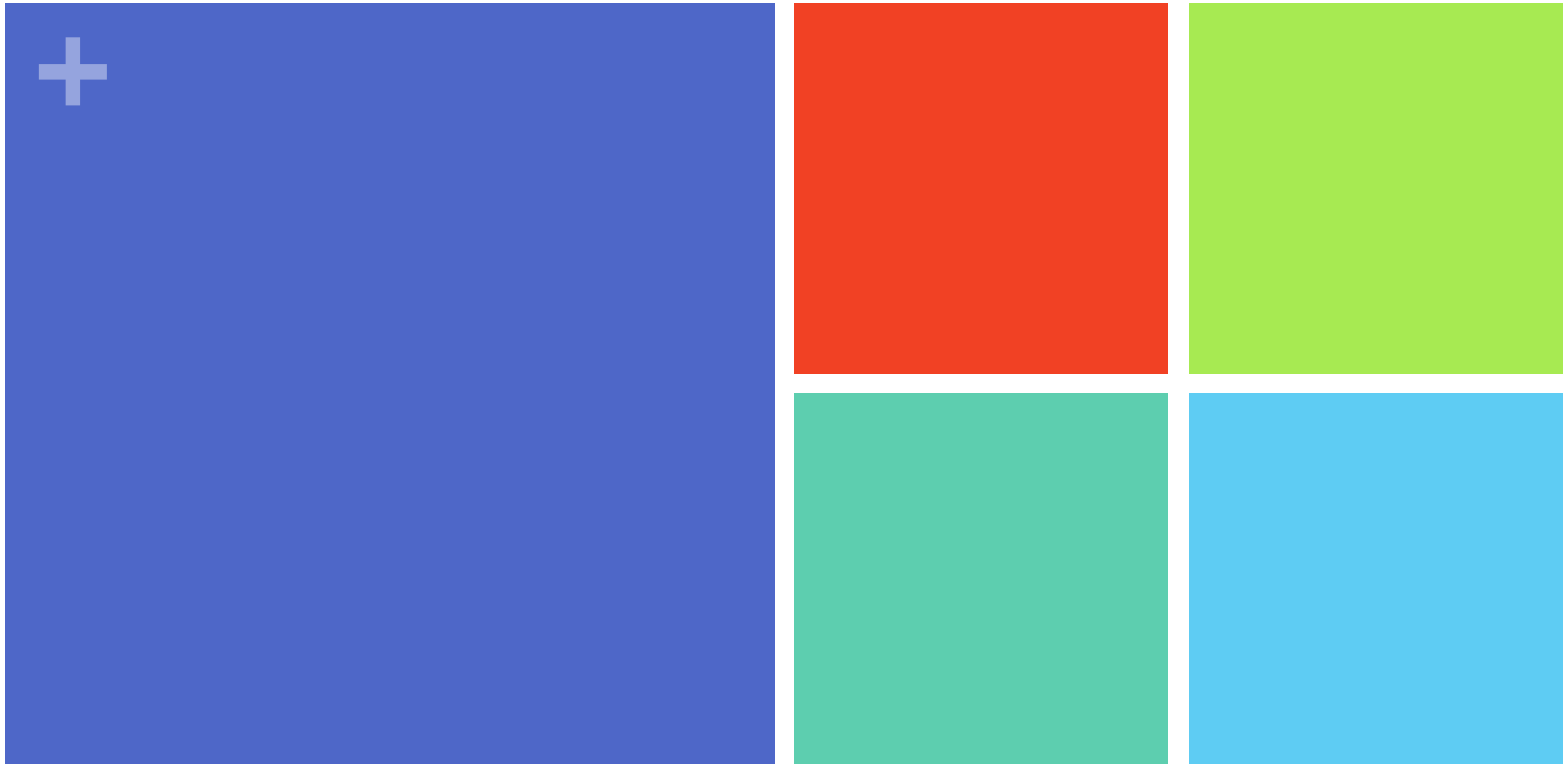
Bài 2.5.2: Xây dựng lớp **Student** mô tả sinh viên với những thành phần sau:

- Thuộc tính: Mã sv, họ tên, điểm Toán, điểm Anh, điểm TB.
- Sử dụng thuộc tính tĩnh để đếm số lượng SV
- Phương thức: Nhập dữ liệu từ bàn phím, hiển thị dữ liệu ra màn hình.
- Có phương thức tính điểm TB theo thang 10
  - Điểm TB = (Toán*2 + Anh) / 3.
- Có phương thức xếp loại học lực của SV theo thang 4
  - $3,6 \leq$ Điểm_TB_thang_4 $\leq 4,0$ ➔ Học lực Xuất sắc
  - $3,2 \leq$ Điểm_TB_thang_4 $< 3,6$ ➔ Học lực Giỏi
  - $2,5 \leq$ Điểm_TB_thang_4 $< 3,2$ ➔ Học lực Khá
  - $2,0 \leq$ Điểm_TB_thang_4 $< 2,5$ ➔ Học lực Trung bình
  - Điểm_TB_thang_4 $< 2,0$ ➔ Học lực Yếu

BÀI TẬP THỰC HÀNH

Bài 2.5.2: Sử dụng lớp **Student** thực hiện các công việc sau:

- Nhập danh sách N (0 < N < 100) sinh viên từ bàn phím. In ra màn hình danh sách sinh viên vừa nhập bao gồm cả thông tin cá nhân, điểm TB và xếp loại học lực
- Tìm tất cả sinh viên tên "Nam" và in ra màn hình
- Đếm và in ra màn hình các sinh viên có điểm Toán > 8 và học lực Khá
- Sắp xếp danh sách N sinh viên theo chiều không giảm của điểm TB và in danh sách sau khi sắp xếp ra màn hình



KIỂU DỮ LIỆU CẤU TRÚC

CẤU TRÚC

- Là một kiểu đơn giản, có kích thước nhỏ, dùng để thay thế cho lớp.
- Các đối tượng được cấp phát trên stack (struct là kiểu giá trị)
- Khai báo

```
[thuộc tính] [bổ sung truy cập] struct <tên cấu trúc> [:<danh sách giao diện>]{
    [<thành viên của cấu trúc>]
}
```

- Ví dụ

```
struct Point {
    int x, y;
    public Point(int x, int y) { this.x = x; this.y = y; }
    public MoveTo (int x, int y) {...}
}
```

CẤU TRÚC

- Cấu trúc có thể được cấp phát với từ khóa new

```
Point p;     // các trường dữ liệu của p chưa được khởi tạo
Point q = new Point();
```

- Các trường dữ liệu không được khởi tạo lúc khai báo

```
struct Point          {
    int x = 0;        // lỗi dịch chương trình
}
```

- Cấu trúc không hỗ trợ việc thừa kế nhưng có thể cài đặt cho giao diện
- Cấu trúc không có bộ hủy và bộ khởi tạo mặc định tùy chọn

HÀM KHỞI TẠO CỦA struct

- Chương trình dịch sẽ tạo ra hàm tạo mặc định, không có tham số cho tất cả các struct (kể cả struct đã khai báo hàm tạo)
- Không cần phải khai báo hàm tạo không có tham số cho struct
- Hàm tạo của struct phải khởi tạo **tất cả** các trường dữ liệu

```
struct Complex {
    double re, im;
    public Complex(double re, double im) {
     this.re = re;
     this.im = im;
    }
    public Complex(double re) : this(re, 0) {}
    ...
}
```

```
Complex c0;        // c0.re và c0.im không được khởi tạo
Complex c1 = new Complex(); // c1.re == 0, c1.im == 0
Complex c2 = new Complex(5);     // c2.re == 5, c2.im == 0
Complex c3 = new Complex(10, 3); // c3.re == 10, c3.im == 3
```

CLASS & STRUCT

Class	Struct
Kiểu tham chiếu (đối tượng được cấp phát trên heap)	Kiểu giá trị (đối tượng được cấp phát trên stack)
Hỗ trợ thừa kế (tất cả các class đều thừa kế từ *object*)	Không hỗ trợ thừa kế (nhưng phải tương thích với *object*)
Có thể cài đặt cho interface	Có thể cài đặt cho interface
Có thể khai báo hàm tạo không có tham số	Không phải khai báo hàm tạo không có tham số
Có thể có hàm huỷ	Không có hàm huỷ

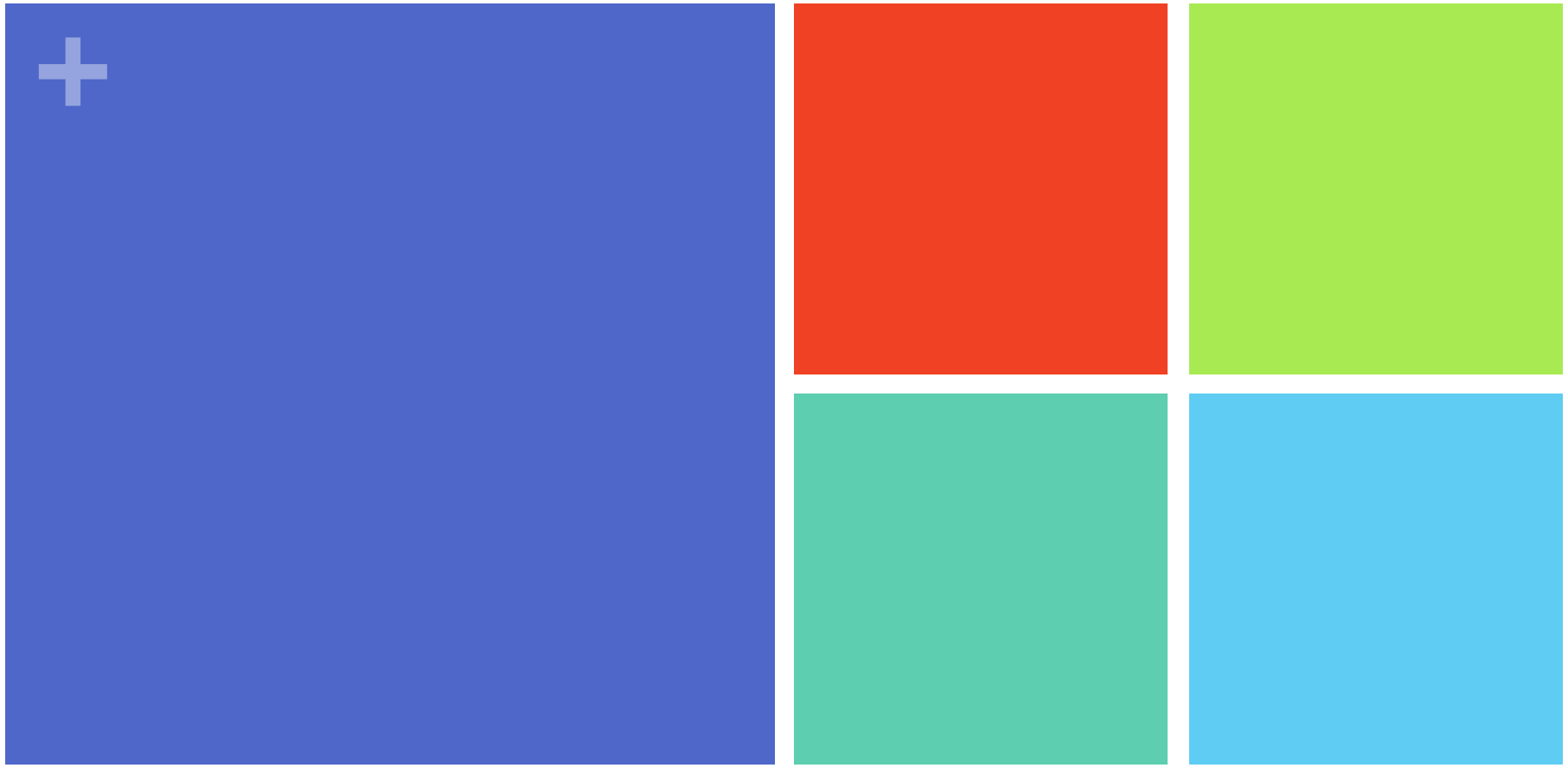
BÀI TẬP THỰC HÀNH

Bài 2.5.3: Sử dụng **Struct** để biểu diễn **Vector** V(a, b) trong hệ tọa độ 2 chiều:

- Thuộc tính: a, b là số nguyên.
- Phương thức get, set giá trị a và b
- Phương thức tính độ dài vector
- Phương thức tính tổng, hiệu của 2 vector

Biết rằng:

- Độ dài Vector V(a, b): $|\vec{v}| = \sqrt{a^2 + b^2}$
- Tổng 2 vector $V_1(a_1, b_1)$ và $V_2(a_2, b_2)$:
  - $\overrightarrow{v_1} + \overrightarrow{v_2} = (a_1 + a_2, b_1 + b_2)$



BOXING VÀ UNBOXING

Class System.Object

- Là class cơ sở của tất cả các kiểu tham chiếu

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

- Có thể được sử dụng như các kiểu chuẩn: *object*

```
object obj;   // chương trình dịch sẽ ánh xạ kiểu object sang kiểu System.Object
```

- Có thể gán đối tượng của kiểu Object cho bất kỳ kiểu tham chiếu nào

```
obj = new Rectangle();
obj = new int[3];
```

- Các phương thức của Object có thể làm việc trên bất kỳ kiểu tham chiếu nào

```
void Push(object x) {...}
Push(new Rectangle());
Push(new int[3]);
```

Boxing và Unboxing

- Các kiểu giá trị (int, struct, enum) tương thích với kiểu *object*

Boxing :

```
object obj = 3;
```

gói giá trị 3 vào một đối tượng trên heap (wrap)

Unboxing :

```
int x = (int) obj;
```

trả giá trị từ đối tượng trên heap (unwrap)

Boxing và Unboxing

```
class Queue  {
          ...
          public void Enqueue(object x) {...}
          public object Dequeue() {...}
          ...
     }
```

Queue có thể sử dụng cả kiểu tham chiếu và kiểu giá trị

```
Queue q = new Queue();

q.Enqueue(new Rectangle());
q.Enqueue(3);

Rectangle r = (Rectangle) q.Dequeue();
int x = (int) q.Dequeue();
```