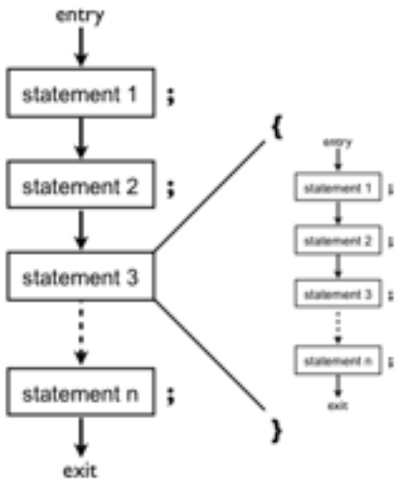
Lớp và đối tượng

v 2.1 - 03/2014

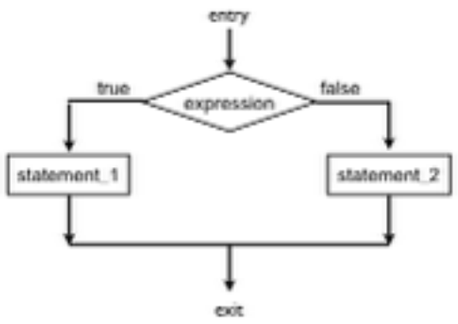
chúng ta đã học...

Programs = Data Structure + Algorithms

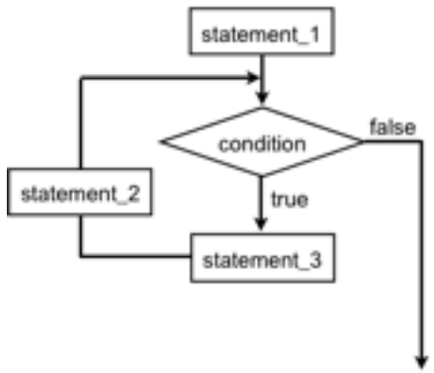
Cấu trúc chương trình



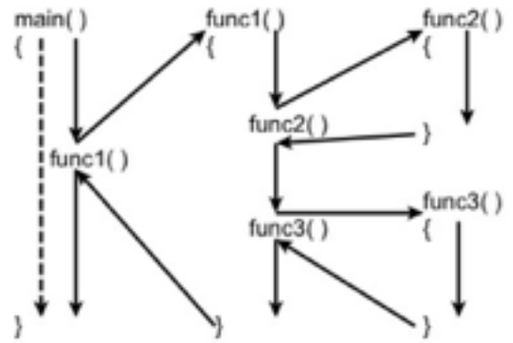
Tuần tự



Rẽ nhánh



Lặp



Hàm

Tổ chức dữ liệu

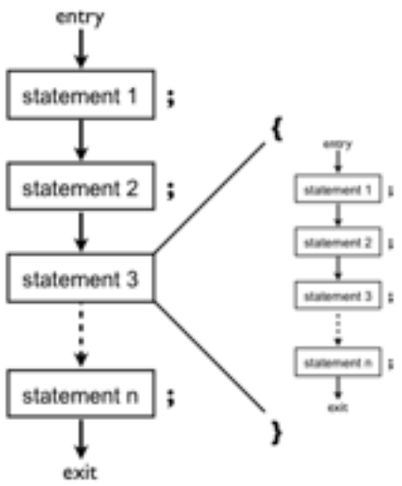
Kiểu dữ liệu

```
bool  char  short  int  long  float  double  string  array
```

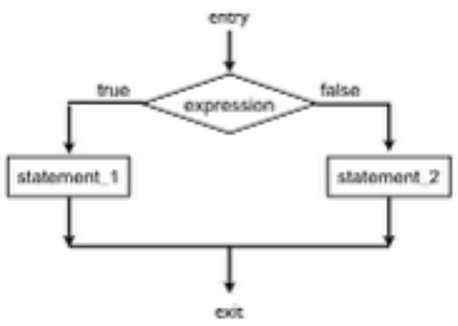
chúng ta sẽ học...

Programs = Object-Oriented Programming

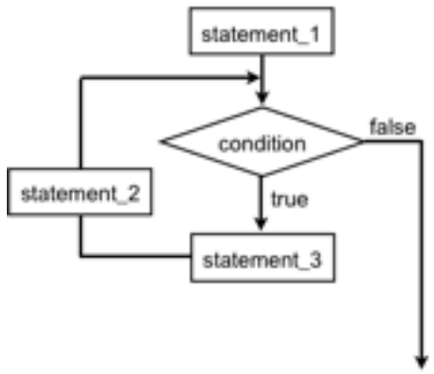
Cấu trúc chương trình



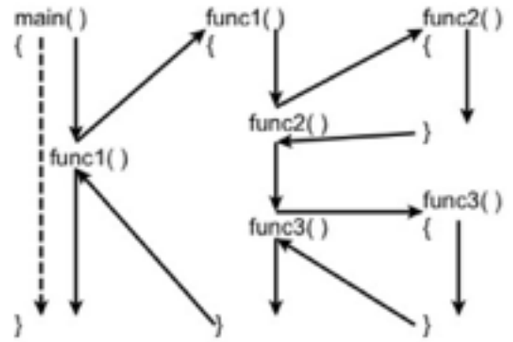
Tuần tự



Rẽ nhánh



Lặp



Hàm

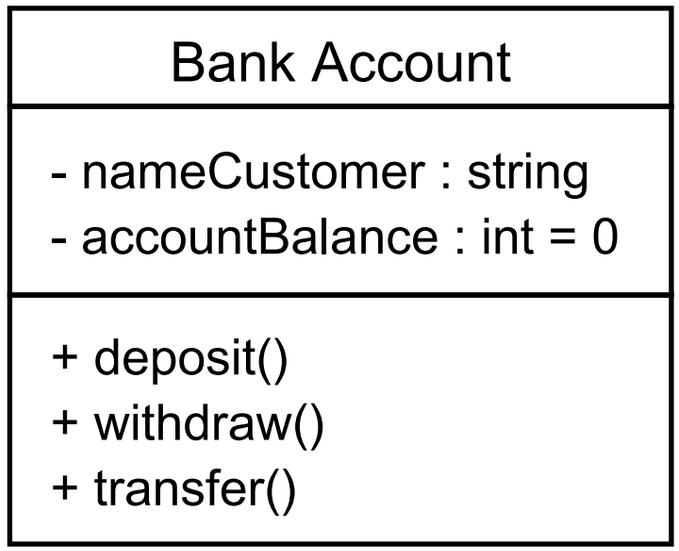
Tổ chức dữ liệu

Kiểu dữ liệu

```
bool  char  short  int  long  float  double  string  array  class
```

cụ thể...

chuyển mô hình



thành code sử dụng C#

Nội dung

1. Mô hình hướng đối tượng
2. Lớp và đối tượng
3. Một số vấn đề khác
    - 3.1. Từ khoá `this`
    - 3.2. Cấu tử
    - 3.3. Các thành phần `static`
    - 3.4. Thuộc tính, thuộc tính tự động
    - 3.5. Biến thành phần chỉ đọc

Mô hình hướng đối tượng

Mô hình hướng đối tượng

- "Object-oriented modeling and design is a new way of ***thinking*** about problems using models ***organized*** around real-world concepts. The fundamental construct is the object, which *combines both data structure and behavior* in a single entity."

  James Rumbaugh, *Object-Oriented Modeling and Design*

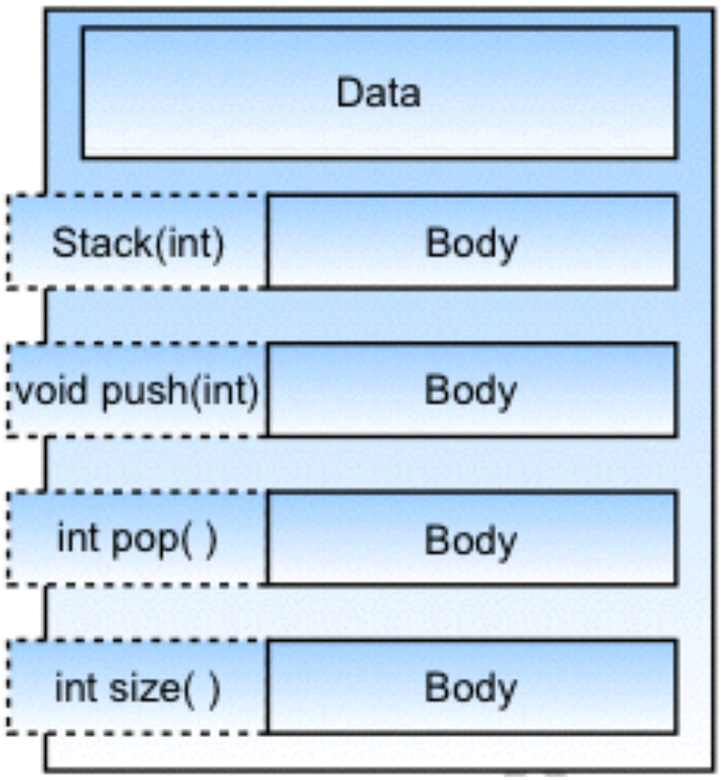
- "Thiết kế và mô hình hoá hướng đối tượng là cách mới để tư duy về các bài toán sử dụng **các mô hình** được *tổ chức* xung quanh **các khái niệm của thế giới thực**. Cấu trúc nền tảng là **đối tượng**, cái mà *kết hợp cả cấu trúc dữ liệu và hành vi* ngay trong cùng một thực thể."

Quy tắc cho thiết kế lớp

- Nghĩ về từ "trách nhiệm"
  - Lớp có trách nhiệm gì trong việc duy trì dữ liệu này ?
  - Lớp có trách nhiệm gì trong việc thực hiện tính toán đó hay cung cấp dịch vụ đó (lớp nào nên định nghĩa hàm đó) ?
- Cài đặt che dấu dữ liệu
  - Thông thường, dùng `private` cho dữ liệu
  - Thông thường, dùng `public` cho các hàm
  - Che dấu các hàm "trợ giúp" (dùng `private`)
  - Chỉ cho phép truy xuất dữ liệu thông qua các hàm `public` (những hàm này là giao diện `public` của lớp)

Giao diện public

- Người sử dụng sử dụng một đối tượng mà không cần biết cách nó làm việc
- Dữ liệu là không *thấy được* từ bên ngoài
- Các thuật toán là cũng không *thấy được* từ bên ngoài
- Chỉ có các đặc tính, giao diện public là đưa ra cho bên ngoài
  - Bao gồm các đặc tính không phải `private`
  - Thường là các hàm thành viên `public`
- Giao diện public là con đường duy nhất để đạt đến sự đóng gói của đối tượng



Hai mặt của lớp

- Thiết kế lớp (người lập trình lớp)
  - Cài đặt một lớp tốt nhất có thể mà không lo lắng về cách nó sẽ được sử dụng
  - Cung cấp dữ liệu thích hợp
  - Cung cấp các hàm thích hợp
  - Cung cấp các đặc tính *rộng* cho các lớp chung, đặc tính *hẹp* cho các lớp chuyên biệt

- Sử dụng lớp (người lập trình ứng dụng)
  - Tập trung vào các giao diện public
    - Lớp đó có thể làm gì, chứ không phải nó làm việc đó thế nào

Lớp và đối tượng

Lớp & đối tượng trong C#

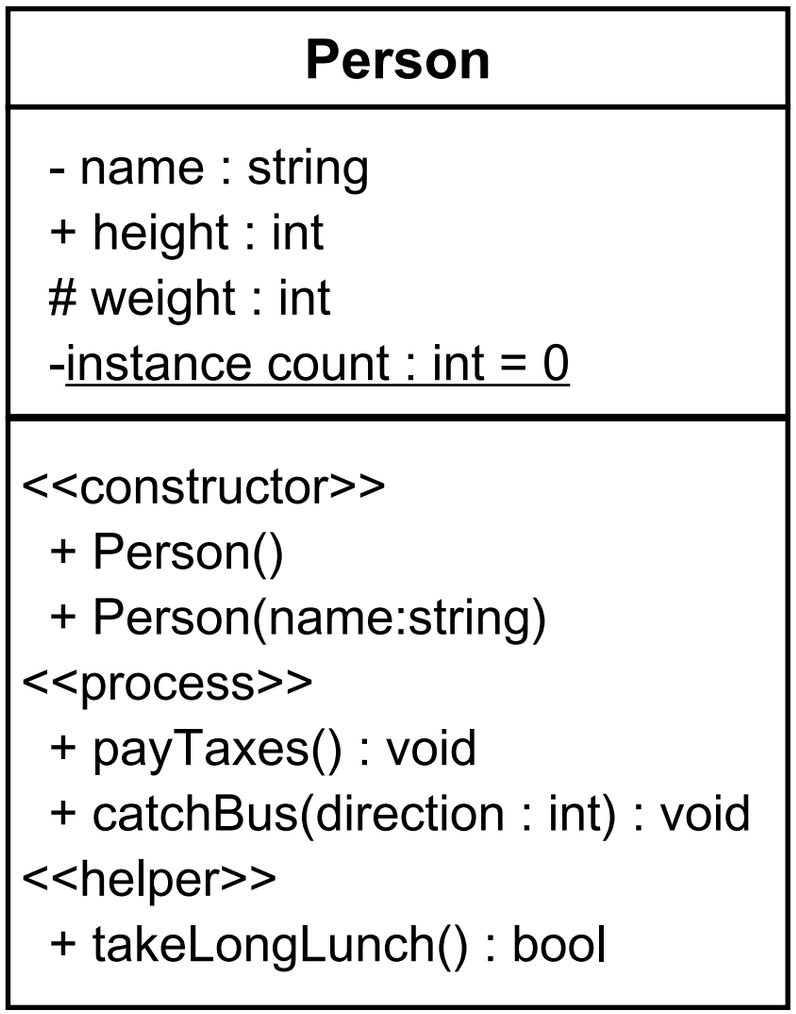
Định nghĩa lớp	Sử dụng *từ khoá* `class` Định nghĩa một *kiểu dữ liệu* mới
Tạo một đối tượng của một lớp	Khai báo một *biến* có kiểu dữ liệu nào đó và ghi dữ liệu vào biến đó Sử dụng từ khoá `new`
Đặc tính	Các *biến thành phần* (gọi tắt là *biến*)
Hành vi	Phương thức / hàm Có hai loại hàm với hai cách triệu gọi khác nhau : • Hàm không tĩnh - Gọi từ đối tượng • Hàm tĩnh (từ khoá `static`) - Gọi từ lớp

```
Bitmap bm = new Bitmap(20, 20);
```

```
Bitmap bm = new Bitmap(20, 20);
bm.Save("bitmap.png");
```

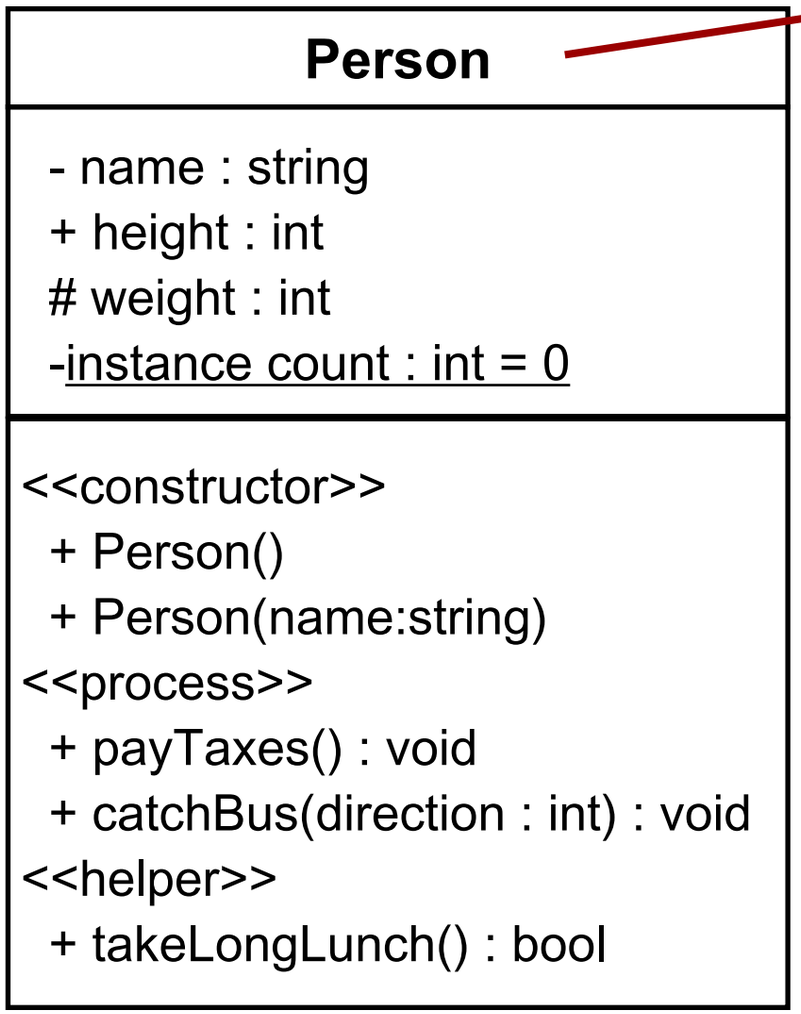
```
Console.Write("Please enter a number :");
```

Lớp UML → Lớp C#



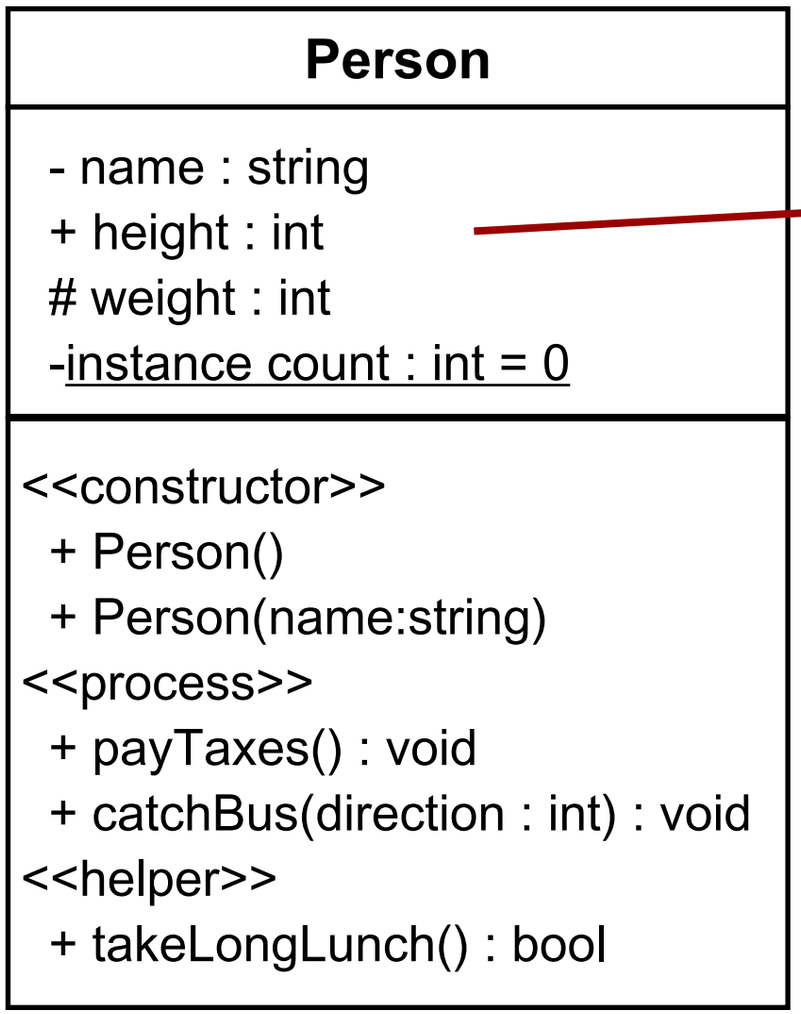
- Tên lớp
  - *ảo*
- Đặc tính (`field`)
  - Biến thành phần
  - Thuộc tính (`property`)
  - Các biến cấp lớp
    - `static`
- Hành vi
  - Cấu tử (`constructor`)
  - Huỷ tử (`destructor`)
  - Hàm thành phần (`method`)

Lớp UML → Lớp C#



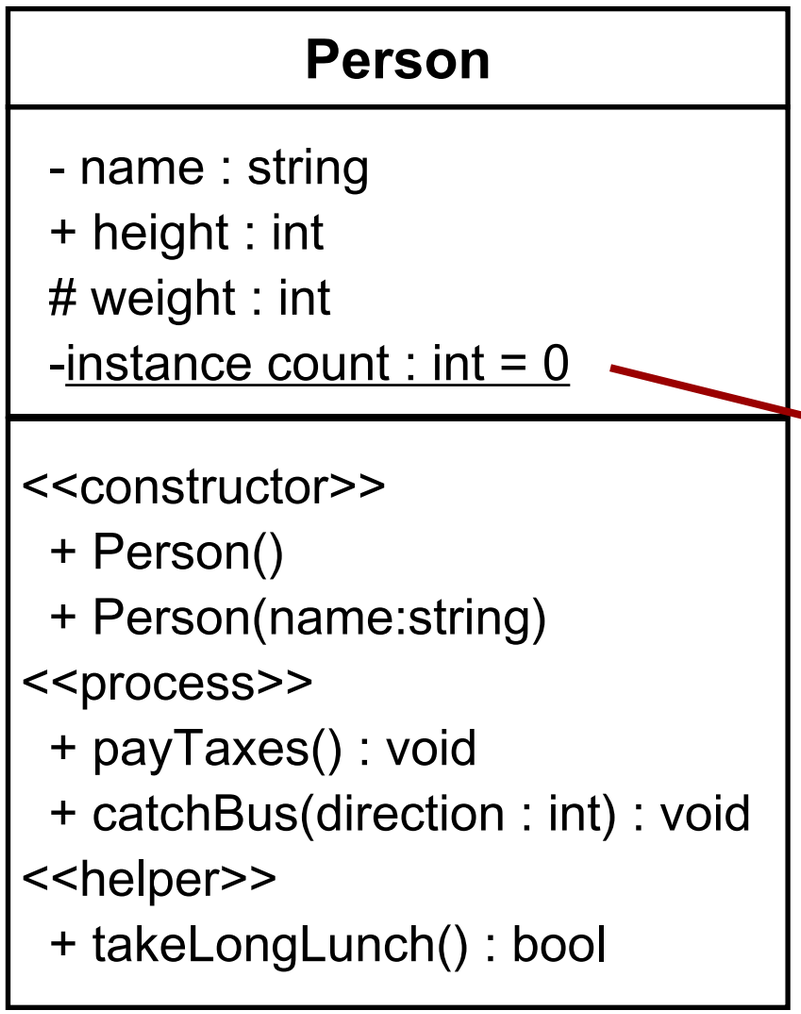
- Tên lớp
  - *ảo*
- Đặc tính (`field`)
  - Biến thành phần
  - Thuộc tính (`property`)
  - Các biến cấp lớp
    - `static`
- Hành vi
  - Cấu tử (`constructor`)
  - Huỷ tử (`destructor`)
  - Hàm thành phần (`method`)

Lớp UML → Lớp C#



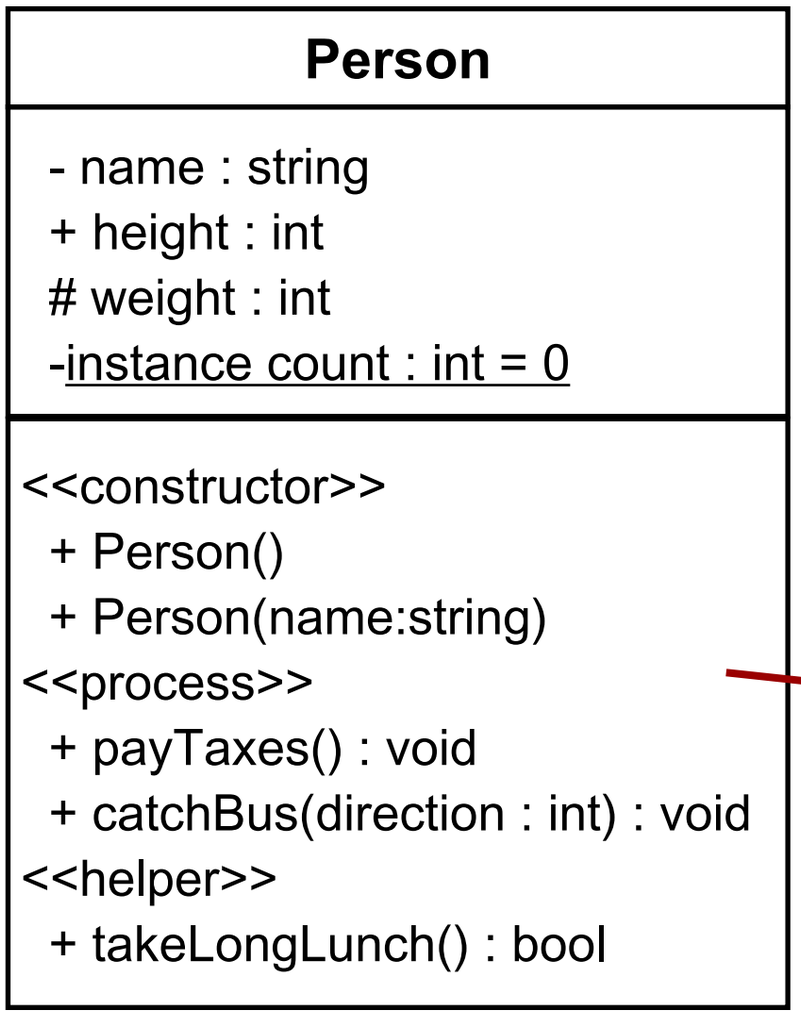
- Tên lớp
  - *ảo*
- Đặc tính (`field`)
  - Biến thành phần
  - Thuộc tính (`property`)
  - Các biến cấp lớp
    - `static`
- Hành vi
  - Cấu tử (`constructor`)
  - Huỷ tử (`destructor`)
  - Hàm thành phần (`method`)

Lớp UML → Lớp C#



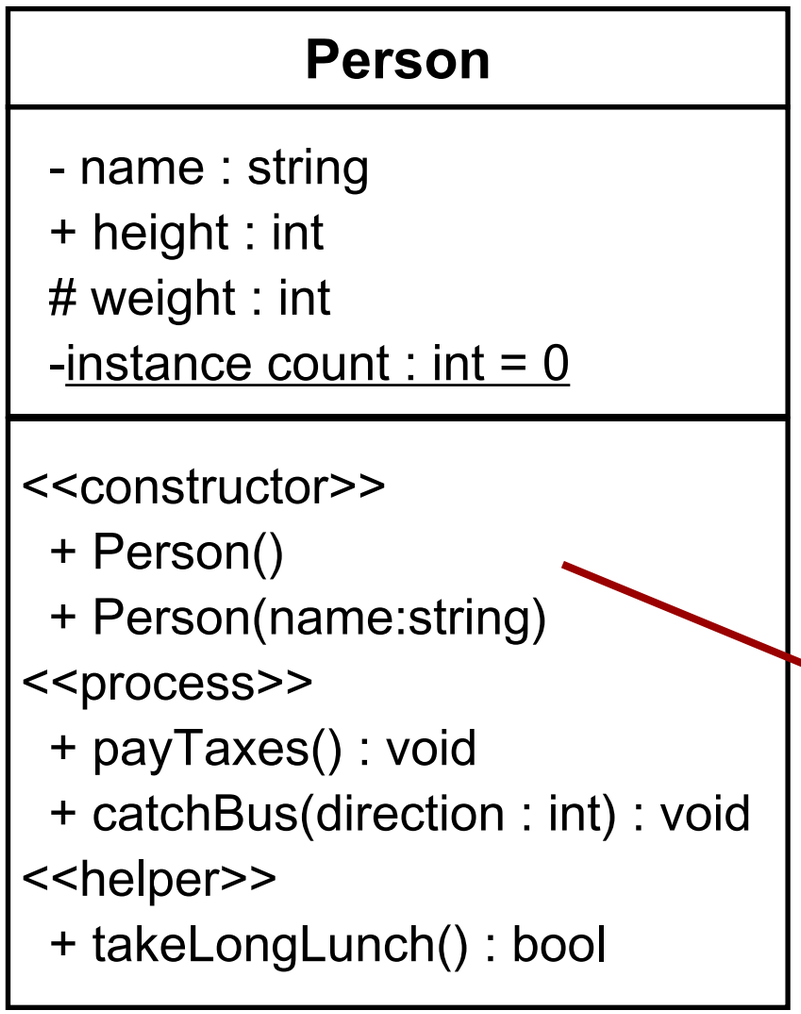
- Tên lớp
  - *ảo*
- Đặc tính (`field`)
  - Biến thành phần
  - Thuộc tính (`property`)
  - Các biến cấp lớp
    - `static`
- Hành vi
  - Cấu tử (`constructor`)
  - Huỷ tử (`destructor`)
  - Hàm thành phần (`method`)

Lớp UML → Lớp C#



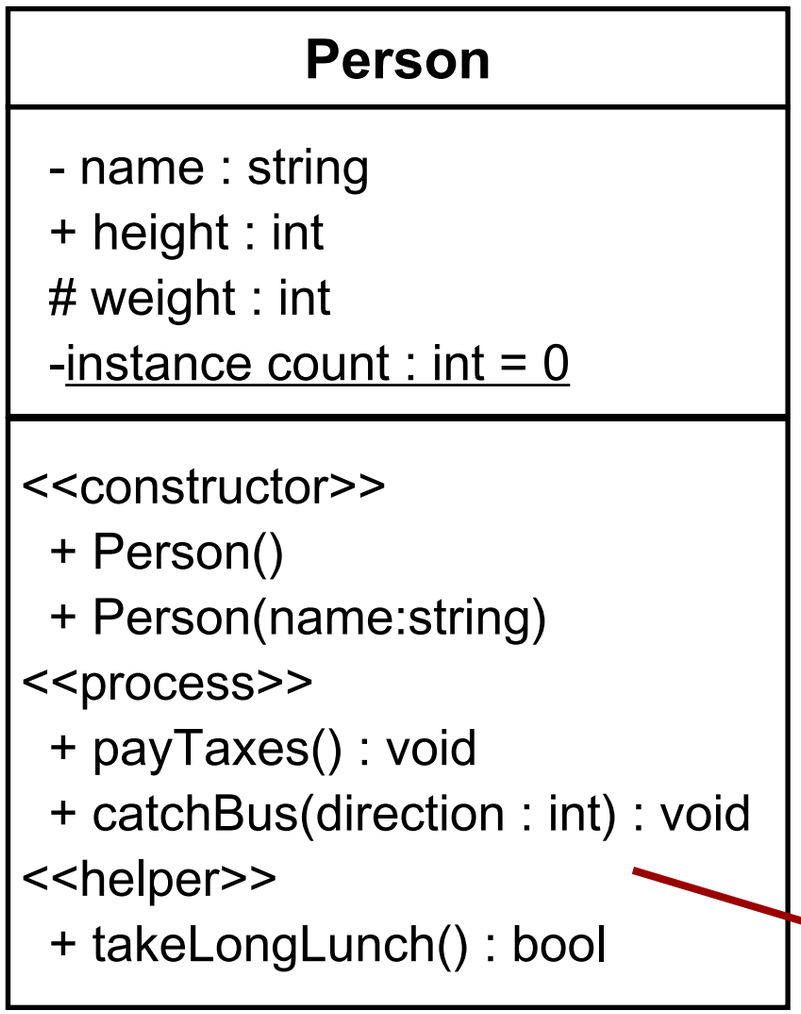
- Tên lớp
  - *ảo*
- Đặc tính (`field`)
  - Biến thành phần
  - Thuộc tính (`property`)
  - Các biến cấp lớp
    - `static`
- Hành vi
  - Cấu tử (`constructor`)
  - Huỷ tử (`destructor`)
  - Hàm thành phần (`method`)

Lớp UML → Lớp C#



- Tên lớp
  - *ảo*
- Đặc tính (`field`)
  - Biến thành phần
  - Thuộc tính (`property`)
  - Các biến cấp lớp
    - `static`
- Hành vi
  - Cấu tử (`constructor`)
  - Huỷ tử (`destructor`)
  - Hàm thành phần (`method`)

Lớp UML → Lớp C#



- Tên lớp
  - *ảo*
- Đặc tính (`field`)
  - Biến thành phần
  - Thuộc tính (`property`)
  - Các biến cấp lớp
    - `static`
- Hành vi
  - Cấu tử (`constructor`)
  - Huỷ tử (`destructor`)
  - Hàm thành phần (`method`)

Ví dụ

Bài toán Mô hình hóa việc tính toán trong sinh vật học

Viết chương trình mô phỏng sự sinh sôi của quần thể virus trong con người theo thời gian. Mỗi tế bào virus tự sinh sôi sau một khoảng thời gian nhất định. Bệnh nhân có thể uống thuốc để kiềm chế quá trình sinh sôi này, và loại bỏ các tế bào virus ra khỏi cơ thể. Tuy nhiên, một số tế bào chống lại thuốc và có thể tiếp tục tồn tại.

Bệnh nhân

Thuộc tính

- số lượng virus
- sự miễn dịch (%)

Hành vi

- uống thuốc

Virus

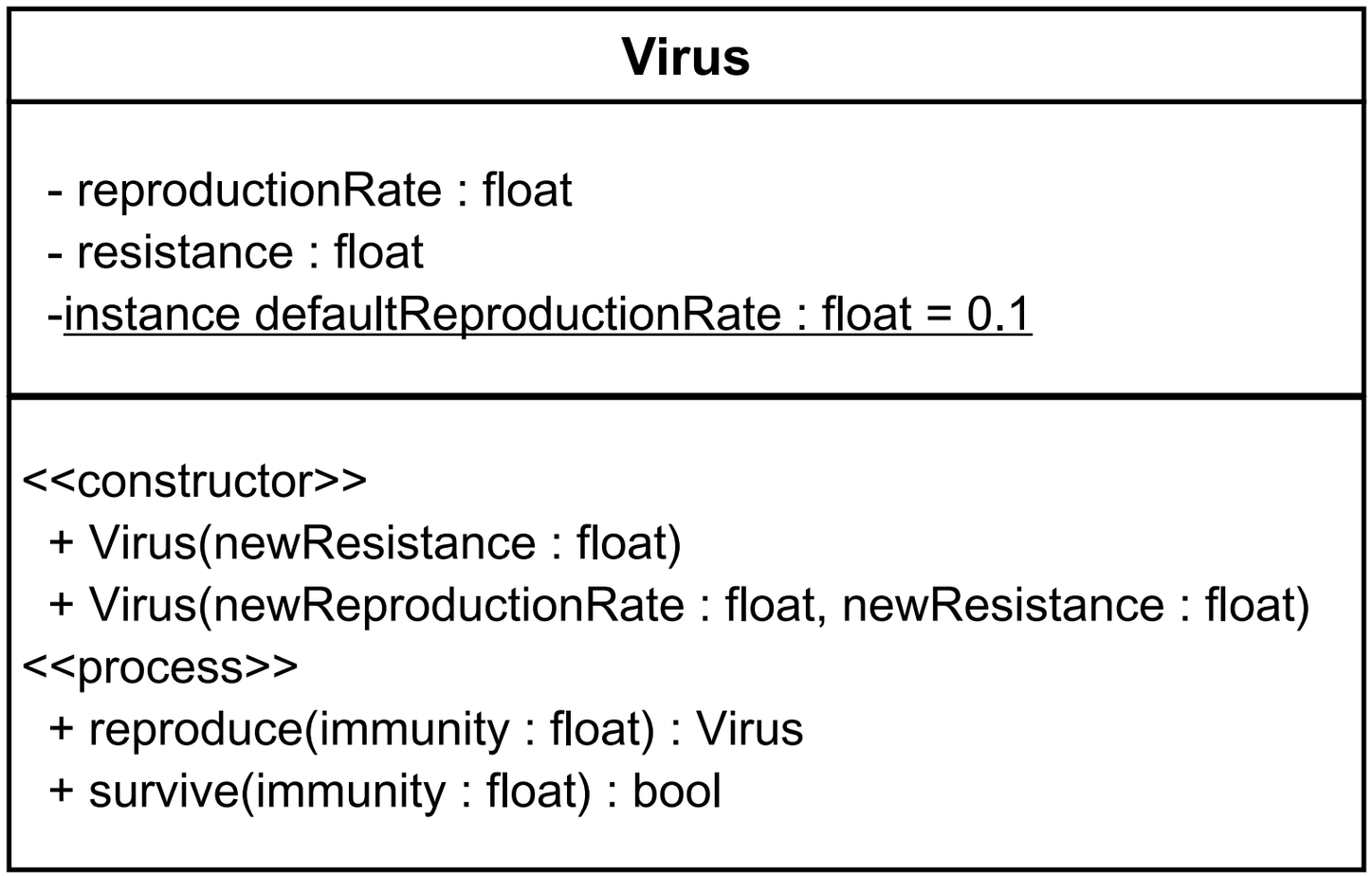
Thuộc tính

- tốc độ sinh sản (%)
- sự kháng thuốc (%)

Hành vi

- sinh sôi
- sống sót

Lớp Virus


Virus
- reproductionRate : float
- resistance : float
-instance defaultReproductionRate : float = 0.1
<<constructor>>
+ Virus(newResistance : float)
+ Virus(newReproductionRate : float, newResistance : float)
<<process>>
+ reproduce(immunity : float) : Virus
+ survive(immunity : float) : bool

Cài đặt lớp `Virus` (1/7)

Virus.cs

```
1  using System;
2
3  namespace virus
4  {
5      class Virus
6      {
7          float reproductionRate; // rate of reproduction, in %
8          float resistance; // resistance against drugs, in %
9          const float defaultReproductionRate = 0.1f;
10
11         public Virus(float newResistance)...
16
17         public Virus(float newReproductionRate, float newResistance)...
22
23         // If this virus cell reproduces,
24         // returns a new offspring with identical genetic info.
25         // Otherwise, returns NULL.
26         public Virus Reproduce(float immunity)...
41
42         // Returns true if this virus cell survives, given the patient's immunity
43         public bool Survive(float immunity)...
50     }
51 }
52
```

Cài đặt lớp `Virus` (2/7)

Virus.cs

```
1  using System;
2
3  namespace virus
4  {
5      class Virus
6      {
7          float reproductionRate; // rate of reproduction, in %
8          float resistance; // resistance against drugs, in %
9          const float defaultReproductionRate = 0.1f;
10
11         public Virus(float newResistance)...
16
17         public Virus(float newRepro
22
23         // If this virus cell repro
24         // returns a new offspring
25         // Otherwise, returns NULL.
26         public Virus Reproduce(floa
41
42         // Returns true if this vi
43         public bool Survive(float
50     }
51 }
52
```

Khai báo mỗi lớp trong file `.cs` riêng

Lớp bắt đầu với tên lớp, theo sau từ khóa `class`

- Tên lớp được đặt theo ký pháp Pascal

Thân lớp đặt trong cặp dấu ngoặc nhọn

Cài đặt lớp `Virus` (3/7)

Virus.cs

```
1  using System;
2
3  namespace virus
4  {
5      class Virus
6      {
7          float reproductionRate; // rate of reproduction, in %
8          float resistance; // resistance against drugs, in %
9          const float defaultReproductionRate = 0.1f;
10
11         public Virus(float newResistance)...
16
17         public Virus(float ne
22
23         // If this virus cell
24         // returns a new offs
25         // Otherwise, returns
26         public Virus Reproduc
41
42         // Returns true if th
43         public bool Survive(float immunity)...
50     }
51 }
52
```

Khai báo các đặc tính của lớp, như khai báo biến

Có thể khởi gán giá trị ban đầu cho các biến này

Cài đặt lớp `Virus` (4/7)

Virus.cs

```
1  using System;
2
3  namespace virus
4  {
5      class Virus
6      {
7          float reproductionRate; // rate of reproduction, in %
8          float resistance; // resistance against drugs, in %
9          const float defaultReproductionRate = 0.1f;
10
11         public Virus(float newResistance)...
16
17         public Virus(fl
22
23         // If this viru
24         // returns a ne
25         // Otherwise, r
26         public Virus Re
41
42         // Returns true
43         public bool Sur
50     }
51 }
52
```

Thành phần dữ liệu hằng

Mặc định là thành phần tĩnh (`static`), nhưng vẫn có thể truy xuất bởi tất cả các thành phần của kiểu

Không thể khởi gán giá trị cho hằng trong cấu tử

Cài đặt lớp `Virus` (5/7)

Virus.cs

```
1  using System;
2
3  namespace virus
4  {
5      class Virus
6      {
7          float reproductionRate; // rate of reproduction, in %
8          float resistance; // resistance against drugs, in %
9          const float defaultReproductionRate = 0.1f;
10
11         public Virus(float newResistance)...
16
17         public Virus(float newReproductionRate, float newResistance)...
22
23         // If this virus cell reproduces
24         // returns a new offspring
25         // Otherwise, returns NULL.
26         public Virus Reproduce(floa
41
42         // Returns true if this virus cell survives, given the patient's immunity
43         public bool Survive(float immunity)...
50     }
51 }
52
```

Cấu tử (`constructor`) cùng tên với lớp
Không có kiểu trả về

Cấu tử - Constructor

- Là hàm thành phần đặc biệt
  - Luôn có cùng tên với lớp chứa nó
  - Không có kiểu trả về (thậm chí cả `void`)
  - Có thể nạp chồng cấu tử
  - Có thể có đối số mặc định
- Được gọi tự động khi một đối tượng được khởi tạo (cấp phát bộ nhớ) sử dụng từ khoá `new`

```
Virus virus = new Virus(0.4f);
```

  - Tất cả các đối tượng của kiểu `class` đều thuộc kiểu tham chiếu
- Được dùng để khởi tạo các biến thành phần của lớp, xin cấp phát vùng nhớ cho các biến thành phần kiểu tham chiếu

Gọi hàm cấu tử

Virus.cs

```
4   {
5       class Virus
6       {
7           float reproductionRate; // rate of reproduction, in %
8           float resistance; // resistance against drugs, in %
9           const float defaultReproductionRate = 0.1f;
10
11          public Virus(float newResistance)...
16
17          public Virus(float newReproductionRate, float newResistance)...
22
```

Program.cs

```
4   {
5       class Program
6       {
7           static void Main(string[] args)
8           {
9               Virus virus = new Virus(0.4f);
10          }
11      }
12  }
```

Gọi hàm cấu tử với số đối số và kiểu đối số tương ứng
Số lượng đối số sẽ phụ thuộc vào các cấu tử mà kiểu đó hỗ trợ

Cài đặt hai cấu tử lớp `Virus`

Virus.cs

```
7       float reproductionRate; // rate of reproduction, in %
8       float resistance; // resistance against drugs, in %
9       const float defaultReproductionRate = 0.1f;
10
11      public Virus(float newResistance)
12      {
13          reproductionRate = defaultReproductionRate;
14          resistance = newResistance;
15      }
16
17      public Virus(float newReproductionRate, float newResistance)
18      {
19          reproductionRate = newReproductionRate;
20          resistance = newResistance;
21      }
22
23      // If this virus cell reproduces
24      // returns a new offspring v
25      // Otherwise, returns NULL.
26      public Virus Reproduce(float
41
42      // Returns true if this viru
43      public bool Survive(float in
50  }
```

Cài đặt cho hai cấu tử

Luôn nhớ khởi tạo cho tất cả các biến thành phần bên trong cấu tử

Cấu tử mặc định

- Mỗi lớp C# có sẵn một cấu tử mặc định (có thể định nghĩa lại khi cần)
- Cấu tử mặc định không có tham số
- Khi tạo đối tượng với lời gọi :
  - Nếu bạn không định nghĩa lại cấu tử mặc định

```
Virus virus = new Virus();
```

  - Sau khi C# cấp phát bộ nhớ cho biến `virus`, cấu tử mặc định đảm bảo tất cả các biến thành phần của lớp `Virus` được khởi gán giá trị sử dụng giá trị mặc định của kiểu dữ liệu của các biến thành phần
- Định nghĩa lại cấu tử mặc định
  - Nhằm khởi tạo giá trị cho các biến thành phần đúng với thực tế
  - Ví dụ : tuổi của nhân viên phải ≥ 18

Chú ý

- Ngay khi bạn cài đặt một cấu tử bất kỳ
  - (kể cả không phải là định nghĩa lại cấu tử mặc định)
  - C# sẽ xoá cấu tử mặc định có sẵn
- Lúc này, nếu bạn vẫn muốn giữ lại một cấu tử mặc định thì nên cài đặt :

```
public Virus() {}
```

Cài đặt lớp `Virus` (6/7)

Virus.cs

```
1  using System;
2
3  namespace virus
4  {
5      class Virus
6      {
7          float reproductionRate; //
8          float resistance; // resist
9          const float defaultReproduc
10
11         public Virus(float newResistance)...
16
17         public Virus(float newReproductionRate, float newResistance)...
22
23         // If this virus cell reproduces,
24         // returns a new offspring with identical genetic info.
25         // Otherwise, returns NULL.
26         public Virus Reproduce(float immunity)...
41
42         // Returns true if this virus cell survives, given the patient's immunity
43         public bool Survive(float immunity)...
50     }
51 }
52
```

Cài đặt các hàm thành phần
Không sử dụng từ khoá `static`

Cài đặt hai hàm thành phần

Virus.cs

```
23          // If this virus cell reproduces,
24          // returns a new offspring with identical genetic info.
25          // Otherwise, returns NULL.
26          public Virus Reproduce(float immunity)
27          {
28              Random r = new Random();
29              float prob = (float) r.NextDouble(); // generate number between 0 and 1
30
31              // If the patient's immunity is too strong, it cannot reproduce
32              if (immunity > prob)
33                  return null;
34
35              // Does the virus reproduce this time?
36              if (prob > reproductionRate)
37                  return null;
38              // No!
39              return new Virus(reproductionRate, resistance);
40          }
41
42          // Returns true if this virus cell survives, given the patient's immunity
43          public bool Survive(float immunity)
44          {
45              // If the patient's immunity is too strong, then this cell cannot survive
46              if (immunity > resistance)
47                  return false;
48              return true;
49          }
```

Có thể truy xuất các biến và hàm thành phần khác trong lớp

Cài đặt lớp `Virus` (7/7)

`private` - chỉ có thể truy xuất bên trong lớp

`public` - có thể truy xuất bởi bất kỳ *ai*

```
1  using System;
2
3  namespace virus
4  {
5      class Virus
6      {
7          float reproductionRate; // rate of reproduction, in %
8          float resistance; // resistance against drugs, in %
9          const float defaultReproductionRate = 0.1f;
10
11         public Virus(float newResistance)...
16
17         public Virus(float newReproductionRate, float newResistance)...
22
23         // If this virus cell reproduces,
24         // returns a new offspring with identical genetic info.
25         // Otherwise, returns NULL.
26         public Virus Reproduce(float immunity)...
41
42         // Returns true if this virus cell survives, given the patient's immunity
43         public bool Survive(float immunity)...
50     }
51 }
52
```

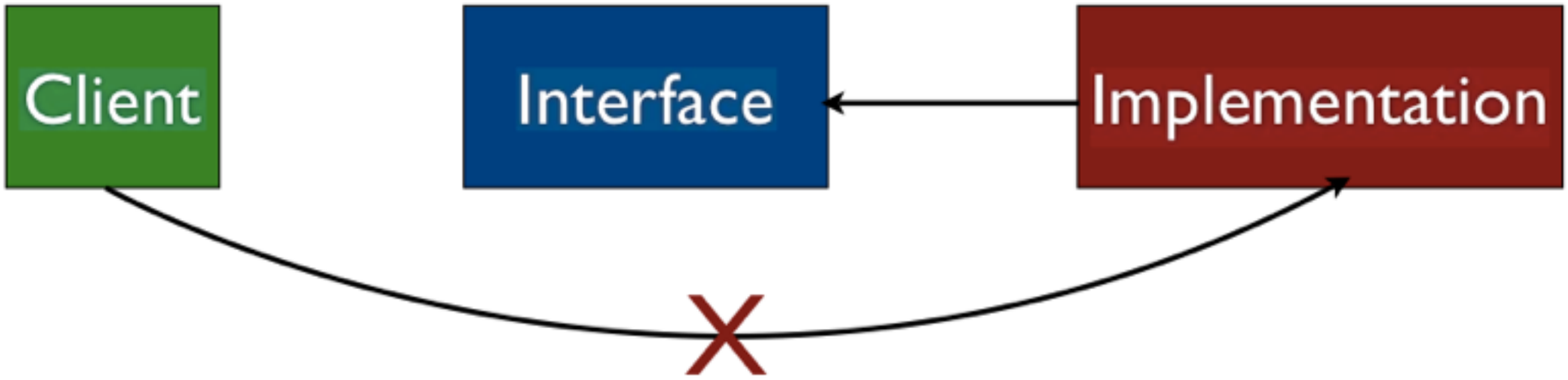
private

public

private vs public

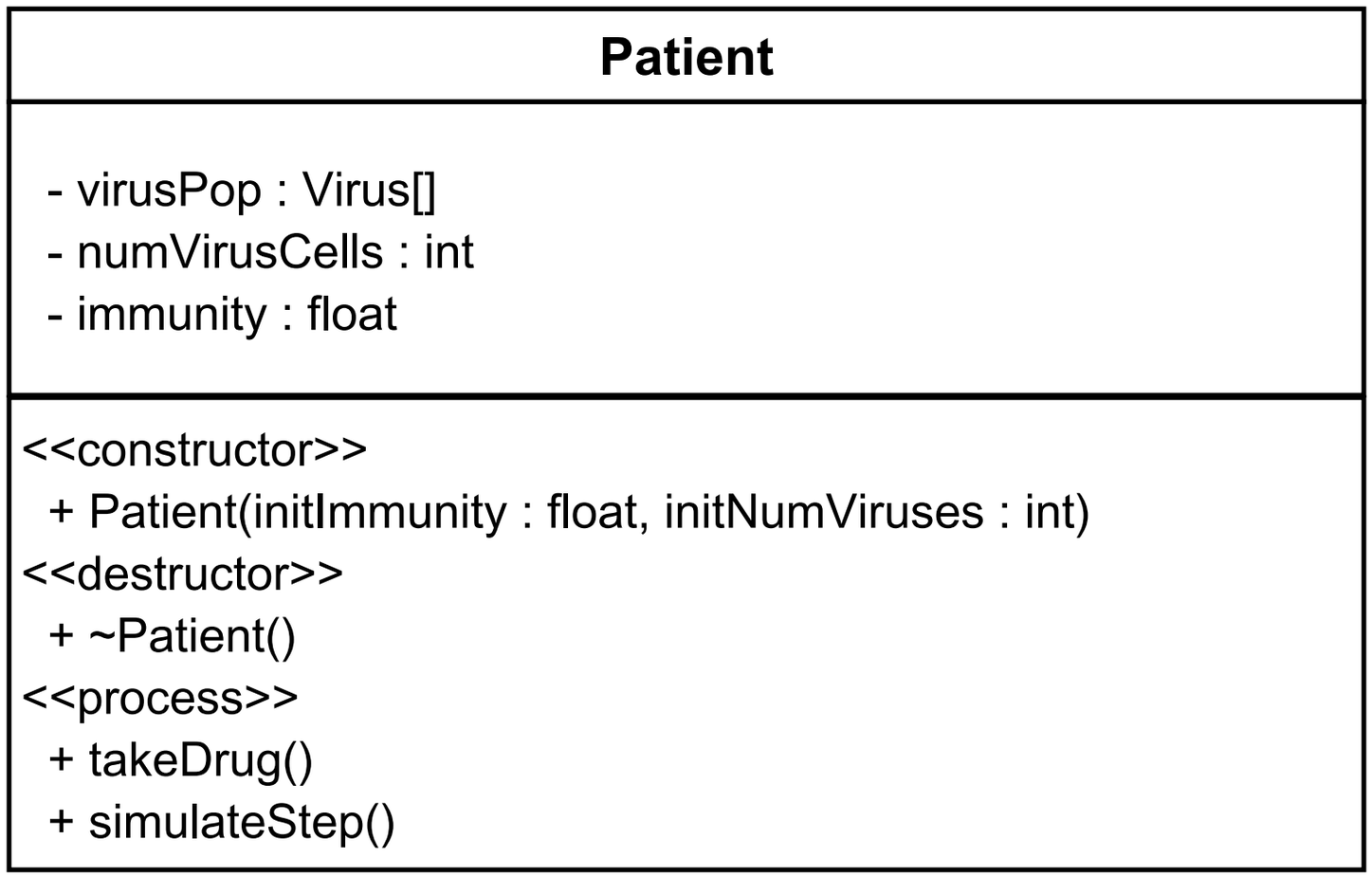
- Giao diện (Interface) - các phần của lớp mà thay đổi không thường xuyên
  - Virus phải được cho phép sinh sôi
- Cài đặt (Implementation) - các phần có thể thay đổi thường xuyên
  - Sự thể hiện của sự kháng thuốc bên trong virus

Bảo vệ thành phần `private`



- Nên
  - Thiết lập biến thành phần là `private`
  - Cực tiểu số hàm thành phần `public`

Lớp Patient


Patient
- virusPop : Virus[]
- numVirusCells : int
- immunity : float
<<constructor>>
+ Patient(initImmunity : float, initNumViruses : int)
<<destructor>>
+ ~Patient()
<<process>>
+ takeDrug()
+ simulateStep()

Cài đặt lớp Patient

Patient.cs

```
1  using System;
2
3  namespace virus
4  {
5      class Patient
6      {
7          Virus[] virusPop;
8          int numVirusCells;
9          float immunity; // degree of immunity, in %
10
11         public Patient(float initImmunity, int initNumVirusCells) ...
27
28         public ~Patient()...
31
32         public void TakeDrug()...
38
39         public void SimulateStep()...
61     }
62 }
63
```

Cài đặt lớp Patient

Patient.cs

```
1  using System;
2
3  namespace virus
4  {
5      class Patient
6      {
7          Virus[] virusPop;
8          int numVirusCells;
9          float immunity; // degree of immunity, in %
10
11         public Patient(float initImmunity, int initNumVirusCells) ...
27
28         public ~Patient()...
31
32         public void TakeDrug()...
38
39         public void SimulateS
61     }
62 }
63
```

virusPop là một mảng các đối tượng kiểu Virus

Hàm hủy tử có khuôn dạng ~<tên lớp>();

Huỷ tử - Destructor

- Là hàm thành phần
  - Không bao giờ có tham số (tức là, không cho phép chồng hàm)
  - Không có kiểu trả về
  - Luôn có cùng tên với lớp chứa nó (cộng thêm dấu ~ ở trước tên hàm)
- Được gọi *tự động* khi một đối tượng vượt quá phạm vi tồn tại của nó hoặc bị xóa
- Được sử dụng để dọn dẹp
  - Hủy cấp phát bộ nhớ

```
virus = null;
```

  - Đóng các file hay xóa các file tạm
  - Ngắt kết nối mạng và cơ sở dữ liệu

Tạo đối tượng của lớp khác

Patient.cs

```
5   class Patient
6   {
7       Virus[] virusPop;
8       int numVirusCells;
9       float immunity; // degree of immunity, in %
10
11      public Patient(float initImmunity, int initNumVirusCells)
12      {
13          float resistance;
14
15          immunity = initImmunity;
16          numVirusCells = initNumVirusCells;
17          virusPop = new Virus[numVirusCells];
18
19          Random r = new Random();
20          for (int i = 0; i < initNumVirusCells; i++) {
21              //randomly generate resistance, between 0.0 and 1.0
22              resistance = (float) r.NextDouble();
23
24              virusPop[i] = new Virus(resistance);
25          }
26      }
27
28      public ~Patient()
```

Sử dụng đ.t đã được cấp phát

Patient.cs

```
40          public void SimulateStep()
41          {
42              Virus virus;
43              bool survived = false;
44
45              for (int i = 0; i < numVirusCells; i++){
46                  virus = virusPop[i];
47
48                  survived = virus.Survive(immunity);
49
50                  if (!survived) {
51                      // delete virus;
52                      // delete virus i
53                      for (int k = i, j = k+1; j < numVirusCells; j++, k++)
54                      {
55                          virusPop[k] = virusPop[j];
56                      }
57
58                      numVirusCells--;
59                  }
60              }
61          }
62
```

Sử dụng toán tử dấu chấm để truy xuất các thành phần `public`

Lớp Programs

Program.cs

```
1  using System;
2
3  namespace virus
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              Patient p = new Patient(0.5f, 10);
10
11             Console.WriteLine("The patient have {0} virus cells.", p.getNumVirusCells());
12
13             p.TakeDrug();
14
15             p.SimulateStep();
16
17             Console.WriteLine("After take drug, the patient have {0} virus cells.", p.getNumViru
18             Console.ReadKey();
19         }
20     }
21 }
22
```

Gọi hàm cấu tử

Patient.cs

```
 9          float immunity; // degree of immunity, in %
10
11          public Patient(float initImmunity, int initNumVirusCells) ...
27
28          public ~Patient()...
31
32          public void TakeDrug()...
38
39          public void SimulateStep()...
61      }
```

Program.cs

```
 5      class Program
 6      {
 7          static void Main(string[] args)
 8          {
 9              Patient p = new Patient(0.5f, 10);
10
11              Console.WriteLine("The patient have {0} virus cells.", p.getNumVirusCells());
12
13              p.TakeDrug();
14
```

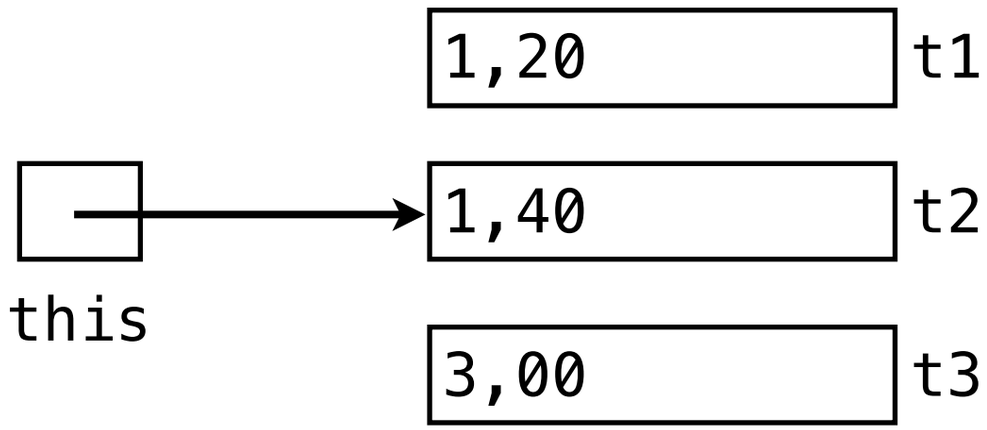
Một số vấn đề khác

Từ khoá `this`

- Cho phép các đối tượng có thể tham chiếu đến chính nó
  - tham chiếu đến chính đối tượng thực hiện lời gọi hàm
- Tham số "bí mật" được truyền cho mỗi hàm thành phần không tĩnh

```
class Time
{
    int hours;
    int minutes;

    public Time(int hours, int minutes)
    {
        this.hours = hours;
        this.minutes = minutes;
    }

    public void Add(Time t)
    {
        int temp = this.minutes + t.minutes;

        if (temp >= 60)
        {
            this.minutes -= 60;
            this.hours++;
        }
        temp = this.hours + t.hours;

        if (temp >= 12)
            this.hours -= 12;
    }
}
```

Từ khoá this




```
static void Main()
{
    Time t1 = new Time(1, 20);
    Time t2 = new Time(1, 40);
    Time t3;

     t3 = t2.Add(t1);
}
```

Gọi cấu tử khác sử dụng `this`

- Thông thường, các cấu tử phải kiểm tra dữ liệu khởi tạo có phù hợp hay không (business rules)
  - Nếu phần kiểm tra này lặp lại trong nhiều cấu tử thì sẽ dẫn đến trùng lặp
- Giải pháp :
  - tách phần kiểm tra này ra một hàm riêng
  - hoặc sử dụng từ khoá this để gọi lại các cấu tử đã có

```
class Motorcycle
{
   ...
   public Motorcycle() {}
   public Motorcycle(int intensity) : this(intensity, “”) {}
   public Motorcycle(string name) : this(0, name) {}
   public Motorcycle(int intensity, string name)
   {
      if (intensity > 10) ...
   }
}
```

Cấu tử và các loại tham số

- Tham số mặc định

```
class Motorcycle
{
   ...
   public Motorcycle(int intensity = 0, string name = “”)
   {
      if (intensity > 10)
         intensity = 10;
      driverIntensity = intensity;
      driverName = name;
   }
   ...
}
```

- Tham số đặt tên
  - có thể bỏ qua các đối số

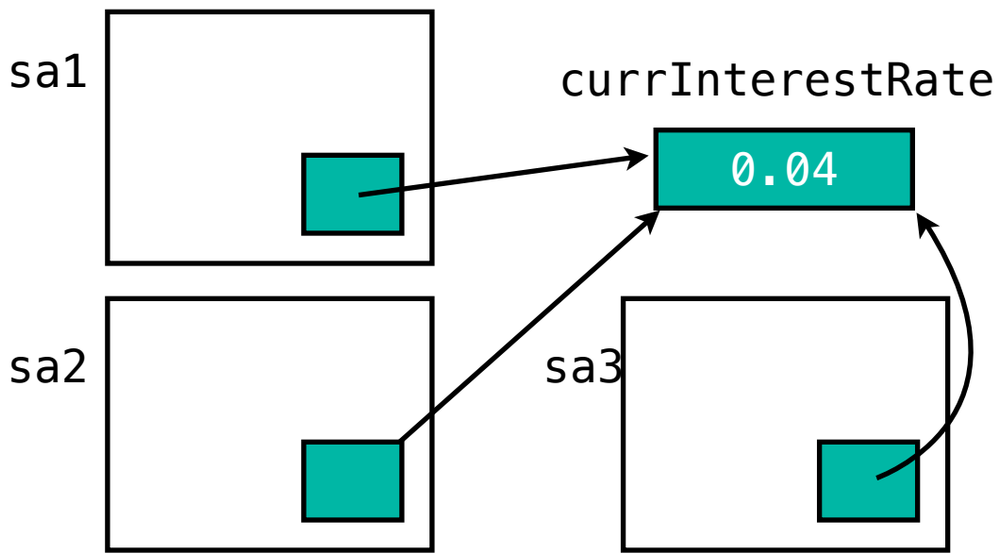
```
Motorcycle m2 = new Motorcycle(name : “Tiny”);
```

Các thành phần `static`

- Các thành phần `static` là các thành phần thuộc cấp lớp
  - Được triệu gọi trực tiếp từ lớp, hay được dùng để chia sẻ cho tất cả các đối tượng
  - Hàm thành phần `static` không có con trỏ this, chỉ có thể truy vấn các thành phần `static` khác

SavingAccount.cs

```
class SavingAccount
{
   double currBalance;
   static double currInterestRate = 0.04;
   public SavingAccount(double balance)
   {
      currBalance = balance;
   }
   public static void
         SetInterestRate(double newRate)
   {
      currInterestRate = newRate;
   }
}
```



Dù có tạo ra bao nhiêu đối tượng mới thì giá trị của biến `static currInterestRate` cũng sẽ không đổi, chỉ thay đổi khi sử dụng hàm `static SetInterestRate`

Cấu tử `static`

- Dùng để khởi gán giá trị cho các biến thành phần `static`

```
class SavingAccount
{
   double currBalance;
   static double currInterestRate;

   public SavingAccount(double balance)
   {
      currInterestRate = 0.04;
      currBalance = balance;
   }
}
```

Giá trị của biến tĩnh currInterestRate sẽ gán lại giá trị 0.04 mỗi khi một đối tượng mới được tạo ra

Giải pháp

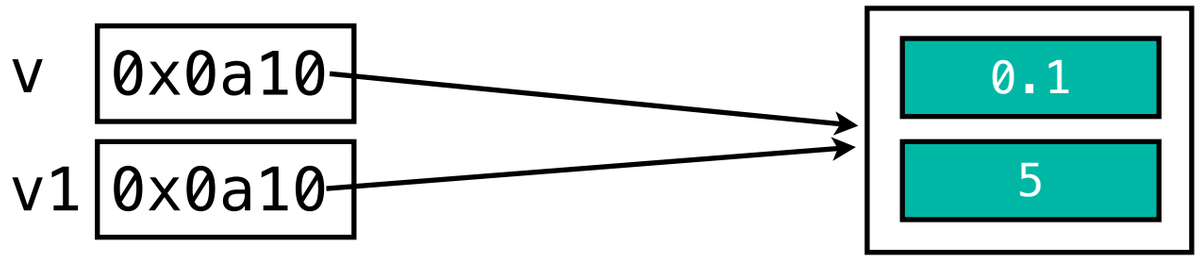
```
class SavingAccount
{
   double currBalance;
   static double currInterestRate;

   public SavingAccount(double balance)
   {
      currBalance = balance;
   }
   public static SavingAccount()
   {
      currInterestRate = 0.04;
   }
}
```

hoặc

```
class SavingAccount
{
   ...
   static double currInterestRate = 0.04;
   ...
```

Phép gán và cấu tử sao chép

```
Virus v = new Virus(0.1, 5);

Virus v1 = v;
```



- C# không cung cấp cấu tử sao chép (chuyển đổi), nên bạn phải cài đặt cấu tử này
  - sao chép từng giá trị của các biến thành phần cho nhau

```
public Virus(Virus virus)
{
    reproductionRate = virus.reproductionRate;
    resistance = virus.resistance;
}
```

Cấu tử `private`

- Là cấu tử đặc biệt, thuộc cấp lớp
  - Dùng cho các lớp chỉ có các thành phần static
    - Ngăn chặn việc tạo ra các đối tượng của các lớp dạng này
    - Ví dụ : lớp Math
  - Thường được cài đặt dưới dạng hàm cấu tử mặc định
    - Để ngăn chặn C# tự động tạo ra cấu tử mặc định

```
public class Counter
{
    private Counter() { }
    public static int currentCount;
    public static int IncrementCount()
    {
        return ++currentCount;
    }
}
```

Truy xuất biến thành phần

- Sử dụng từ khoá `public` cho các biến thành phần
  - Vi phạm tính chất an toàn dữ liệu
    - Thay đổi dữ liệu mà chưa qua kiểm tra tính hợp lý dữ liệu (business rules) đã được cài đặt sẵn trong lớp
- Sử dụng từ khoá `private` cho các biến thành phần
  - Định nghĩa một cặp hàm lấy dữ liệu (`get`) và gán dữ liệu (`set`)
  - Định nghĩa các thuộc tính (`.NET Property`)

Hàm get và hàm set

```
class Employee
{
    private string empName;

    public string GetName()
    {
        return empName;
    }

    public void SetName(string name)
    {
        if (value.Length > 15)
                empName = “”;
            else
                empName = name;
    }
}
```

Cần 2 hàm cho mỗi biến thành phần

Sử dụng thuộc tính (Property)

```
class Employee
{
    private string empName;

    public string Name
    {
        get { return empName; }
        set
        {
            if (value.Length > 15)
                empName = “”;
            else
                empName = value;
        }
    }
}
```

- Không có cặp dấu ngoặt, không có tham số
- Tên của thuộc tính được đặt theo ký pháp Pascal
- Kiểu trả về thể hiện kiểu dữ liệu mà thuộc tính đóng gói
- Token value để thể hiện giá trị truyền vào
- Vẫn có thể thực hiện kiểm tra tính hợp lý dữ liệu trước khi thực hiện gán dữ liệu

```
Employee joe = new Employee();
joe.Name = “Joe”;

joe.Age++;
```

Lợi thế của thuộc tính

Thuộc tính và cấu tử

- Cấu tử thường có các kiểm tra hợp lý dữ liệu
  - Có thể tận dụng các thuộc tính

```
public Employee(string name, int age, int id, float pay)
{
    Name = name;
    Age = age;
    ID = id;
    Pay = pay;
}
```

Một số vấn đề của thuộc tính

- Điều khiển khả năng truy xuất của thuộc tính

```
public string SocialSecurityNumber
{
    get { return empSSN; }
    protected set { empSSN = value; }
}
```

- Thuộc tính chỉ đọc và chỉ viết

```
public string SocialSecurityNumber
{
    get { return empSSN; }
}
```

- Thuộc tính `static`

```
class Employee
{
    private static string companyName;
    public static string CompanyName
    {
        get { return companyName; }
        set { companyName = value; }
    }
}
```

Thuộc tính tự động

- Đối với các biến thành viên không cần kiểm tra tính hợp lệ dữ liệu (business rules) thì có thể dùng cú pháp thuộc tính tự động

```
class Car
{
    public string PetName { get; set; }
    public int Speed { get; set; }
    public string Color { get; set; }
}
```

- Trình biên dịch tự động định nghĩa các biến thành phần ẩn tương ứng tại thời điểm biên dịch
  - Không thể sử dụng trực tiếp như các biến thành phần
- Thuộc tính tự động không hỗ trợ chỉ đọc và chỉ ghi, nhưng có thể giới hạn khả năng truy xuất

```
public string OtherProperty{ get; protected set; }
```

T.tính tự động và g.trị mặc định

- Mặc định, trình biên dịch sẽ gán giá trị mặc định cho các biến thành phần ẩn
  - Lỗi với các kiểu tham chiếu

```
class Garage
{
    public int NumberOfCars { get; set; }
    public Car MyAuto { get; set; }
}
```

```
Garage g = new Garage(); // cấu tử mặc định được gọi

// Lỗi vì MyAuto là null
Console.WriteLine(g.MyAuto.PetName);
```

- Cần định nghĩa lại cấu tử mặc định

Cú pháp khởi gán đối tượng

- Cú pháp cho phép khởi gán giá trị cho các thuộc tính mà không phụ thuộc vào các cấu tử đã được cài đặt
- Điều kiện : Lớp có các biến `public` hoặc thuộc tính `public`

```
class Point
{
    public int X { get; set; }
    public int Y { get; set; }

    public Point(int xVal, int yVal)
    {
        X = xVal;
        Y = yVal;
    }

    public Point() {}
}
```

Cú pháp khởi gán đối tượng

```
static void Main()
{
    Point firstPoint = new Point();
    firstPoint.X = 10;
    firstPoint.Y = 10;

    Point anotherPoint = new Point(20, 20);
    // gọi không tường minh cấu tử mặc định
    Point finalPoint = new Point { X = 30, Y = 30 };
}
```

- Gọi tường minh cấu tử mặc định

```
Point finalPoint = new Point() { X = 30, Y = 30 };
```

- cũng có thể gọi

```
Point finalPoint = new Point(10, 10) { X = 30, Y = 30 };
```

Biến thành phần chỉ đọc

- Sử dụng từ khoá `readonly`

```
class MyMathClass
{
    public readonly double PI;
    public MyMathClass() {
        PI = 3.14;
    }
}
```

- Giống hằng : giá trị không thể thay đổi sau khi khởi gán
- Khác hằng : giá trị được gán tại thời gian chạy (`runtime`), nên có thể gán giá trị trong hàm cấu tử (và chỉ cấu tử)
- Sử dụng : khi giá trị hằng số này bạn không biết trước, mà chỉ có thể xác định tại thời gian chạy (vd : đọc từ file)
- Biến thành phần chỉ đọc không phải là `static`
  - nếu muốn nó là `static` thì phải thêm tường minh từ khoá `static`

Cảm ơn sự chú ý

Câu hỏi ?