MISA®
TIN CẬY - TIỆN ÍCH - TẬN TÌNH

Giới thiệu ngôn ngữ lập trình C#

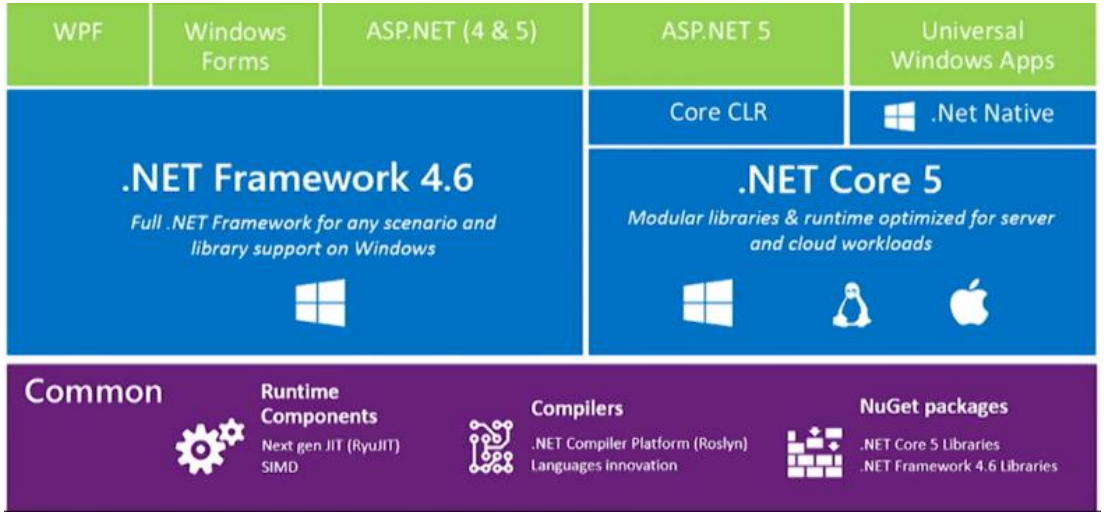
Nguyễn Thanh Tùng – CTO

Nội dung

1. Định nghĩa class
2. Mức độ truy cập (access modifier)
3. Hàm khởi tạo
4. Trường, hằng số và thuộc tính
5. Thành phần tĩnh (Static)
6. Flow control: vòng lặp, exception handling…

.NET Framework và C#

- .NET Framework hỗ trợ xây dựng tất cả các loại ứng dụng từ Desktop -> Web -> Mobile
- Tất cả ứng dụng .NET đều là hướng đối tượng
- C# là một ngôn ngữ trên .NET, ngoài ra còn có rất nhiều ngôn ngữ khác: VB.NET, VC++, J#, F#...

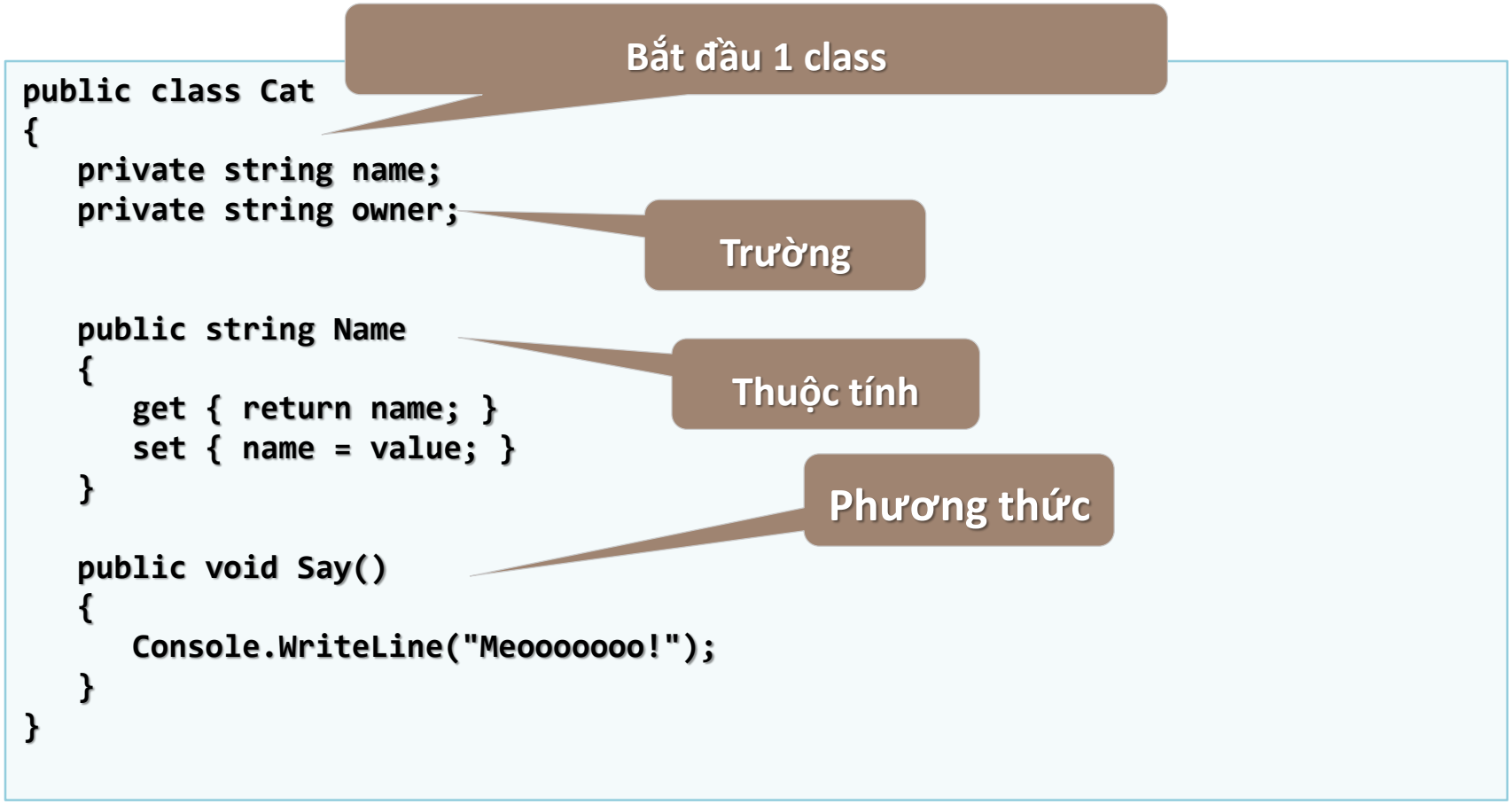


Lớp (Class)

Lớp

- Lớp mô hình hóa các đối tượng trong thực tế, bao gồm:
  - Thuộc tính (trạng thái, trường)
  - Hành vi (phương thức, hành động)
- Lớp mô tả cấu trúc của các đối tượng
  - 1 đối tượng là một thể hiện cụ thể của 1 lớp

Demo tạo 1 lớp đơn giản



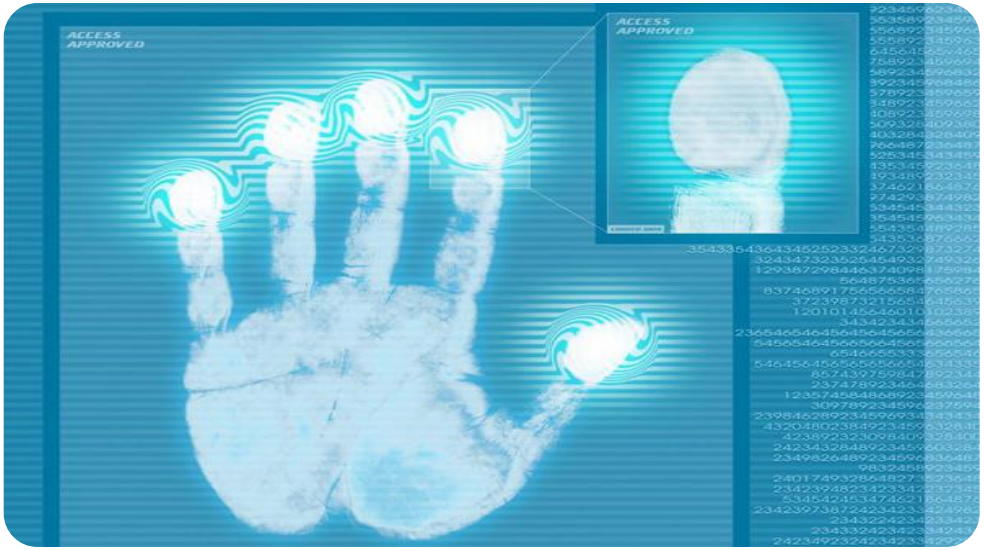

```
public class Cat
{
    private string name;
    private string owner;


    public string Name
    {
        get { return name; }
        set { name = value; }
    }

    public void Say()
    {
        Console.WriteLine("Meooooooo!");
    }
}
```

Mức truy cập

- Public, Private, Protected, Internal



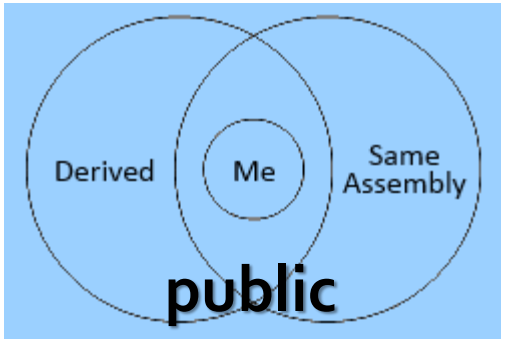
Mức truy cập

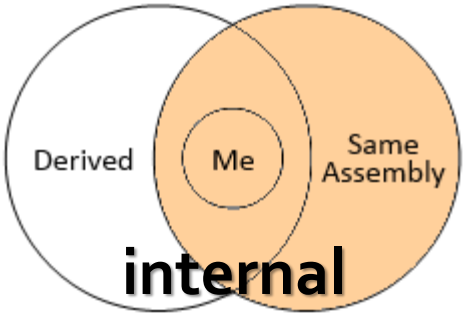
- Áp dụng cho lớp hoặc các thành phần của lớp để:

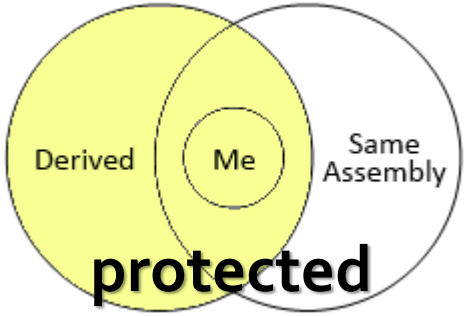
  Giới hạn truy cập tới chúng từ các lớp khác

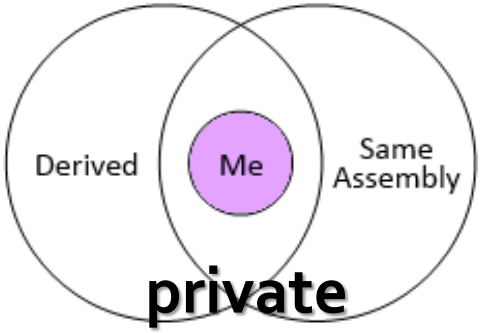
  – `public` – Có thể truy cập từ các lớp khác
  – `protected` – Chỉ truy cập được trong chính lớp đó và các lớp kế thừa nó
  – `private` – Chỉ truy cập được trong chính lớp đó
  – `internal` – Truy cập được trong cùng 1 assembly (namespace hay trong cùng file .dll, .exe)

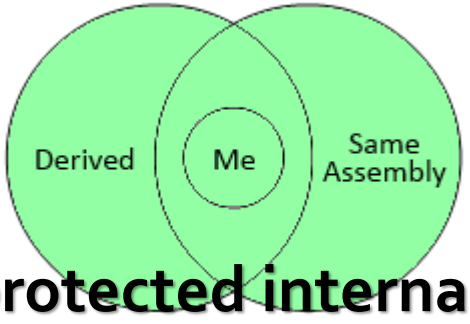
Mức truy cập


Derived
Me
Same Assembly
public


Derived
Me
Same Assembly
internal


Derived
Me
Same Assembly
protected


Derived
Me
Same Assembly
private


Derived
Me
Same Assembly
protected internal

Demo sử dụng lớp

- Tạo một đối tượng của lớp
- Truy cập/thay đổi thuộc tính
- Gọi hàm

Hàm khởi tạo

Hàm khởi tạo là gì?

- Là một phương thức đặc biệt
  - Được gọi khi khởi tạo một object của class
  - Thường dùng để khởi tạo các trường cho object đó
- Hàm khởi tạo có cùng tên như class
  - Không có kiểu trả về
  - Có thể có tham số
  - Các mức truy xuất là: `private`, `protected`, `internal`, `public`

Định nghĩa hàm khởi tạo

- **Không tham số:**

```
public class Cat
{
    private string name;
    private string owner;

    public Cat()
    {
        this.name = “Noname”;
        this.owner = “Noowner”;
    }
}
```

Có tham số

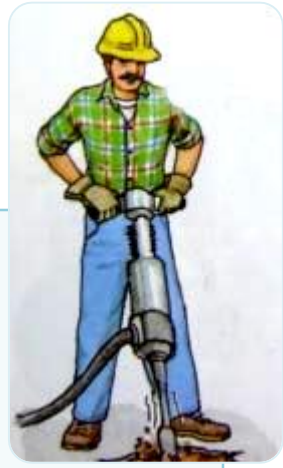
```
public class Cat
{
   private string name;
   private string owner;

   public Cat()
   {
      this.name = “Noname”;
      this.owner = “Noowner”;
   }

   public Cat(string name, string owner)
   {
      this.name = name;
      this.owner = owner;
   }
}
```

Gọi hàm khởi tạo nối nhau

- Tận dụng (reuse) hàm khởi tạo đã có

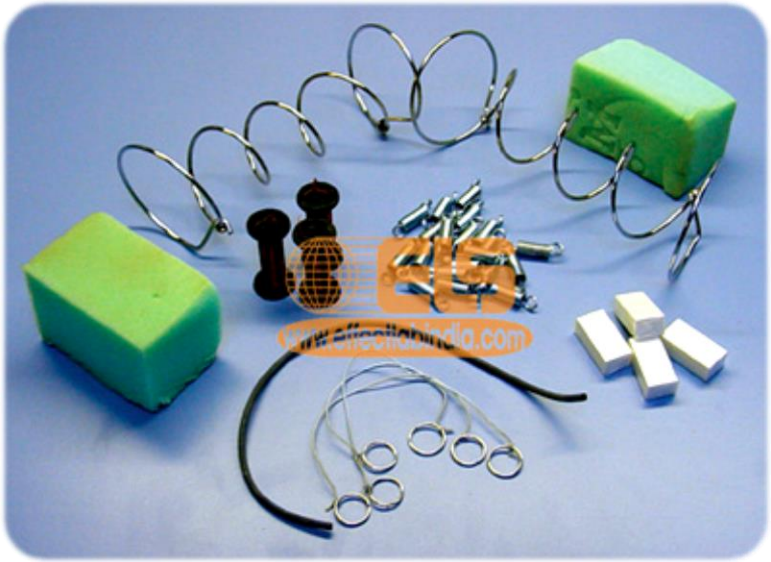


```
public class Cat
{
   private string name;
   private string owner;

   public Cat() : this("Noname", "Noowner") // Reuse constructor
   {
   }

   public Cat(string name, string owner)
   {
      this.name = name;
      this.owner = owner;
   }
}
```

Trường, Hằng số và Thuộc tính



Trường

- Chứa giá trị cho đối tượng
- Có thể là bất kì kiểu gì
- Có phạm vi

```
class Student
{
    private string firstName;
    private string lastName;
    private int course = 1;
    private string speciality;
    protected Course[] coursesTaken;
    private string remarks = "(no remarks)";
}
```

Hằng số

- Được định nghĩa như trường nhưng:
  - Thêm từ khóa `const`
  - Phải khởi tạo ở thời điểm định nghĩa
  - Giá trị không thể thay đổi khi chạy

```
public class MathConstants
{
    public const string PI_SYMBOL = "π";
    public const double PI = 3.14159265358979323385;
    public const double E = 2.71828182845904523354;
    public const double LN10 = 2.30258509299405;
    public const double LN2 = 0.693147180559945;
}
```

Trường chỉ đọc (Read-Only Fields)

- Được khởi tạo ở lúc định nghĩa hoặc trong hàm khởi tạo
  - Sau đó không thể chỉnh sửa
- Định nghĩa với từ khóa `readonly`

```
public class ReadOnly
{
    private readonly int size;
    public ReadOnly(int Size)
    {
      size = Size; // can not be further modified!
    }
}
```

Thuộc tính

- Thể hiện dữ liệu của đối tượng ra bên ngoài
- Kiểm soát dữ liệu được sửa đổi như thế nào:
  - Read-only
  - Write-only
  - Read and write

```
private string _name;
public string Name
{
    get { return _name; }
    set { _name = value; }
}
```

Phương thức (Method)

- Có 2 loại
  - Thực hiện nhưng không trả về giá trị bằng khai báo **void**
  - Thực hiện có trả về giá trị sử dụng **return**

```
public class Cat
{
        public void SayWithName()
        {
            Console.WriteLine(String.Format("{0} say: My owner is {1}!", _name, _owner));
        }

        public string GetName()
        {
            string fullName = "My name is: " + _name;
            return fullName;
        }
}
```

Thành phần tĩnh (static)

Static

- Static gắn với lớp (class/type) chứ không gắn với đối tượng, khai báo bằng từ khóa `static`
- Dùng khi cần khai báo các thuộc tính hoặc phương thức chung cho cả một lớp

Static vs. Non-Static

Static (Class)	Non-Static (Object)
Gắn với class, không gắn với object	Ngược lại, gắn với object
Khởi tạo ngay trước khi kiểu được dùng lần đầu tiên	Khởi tạo khi gọi hàm khởi tạo đối tượng

Static – Example

```
public class SqrtPrecalculated
{
   public const int MAX_VALUE = 10000;

   // Static field
    public static int Max_SqrtNumber = MAX_VALUE;
   private static int[] sqrtValues;

   static SqrtPrecalculated()
   {
      sqrtValues = new int[MAX_VALUE + 1];
      for (int i = 0; i < sqrtValues.Length; i++)
      {
         sqrtValues[i] = (int)Math.Sqrt(i);
      }
   }
     // Static method
   public static int GetSqrt(int value)
   {
      return sqrtValues[value];
   }
}
```

Flow control

Việc chuyển kiểu

- Khi chuyển từ kiểu "Nhỏ hơn" về kiểu "Lớn hơn" thì không cần chỉ rõ kiểu. Ngược lại sẽ phải chỉ định rõ kiểu chuyển như sau

```
int i = 5;
long l = i;
```

```
long l = 5;
int i = (int) l;
```

Vòng lặp While(…)

- Lặp cho đến khi điều kiện còn đúng

```
int counter = 0;
while (counter < 10)
{
    Console.WriteLine("Number : {0}", counter);
    counter++;
}
```

```
Number : 0
Number : 1
Number : 2
Number : 3
Number : 4
Number : 5
Number : 6
Number : 7
Number : 8
Number : 9
Press any key to continue_
```

VD While loop

- Tính và in ra tổng của N số tự nhiên đầu tiên

Vòng lặp for…

- Lặp theo một số lần nhất định. VD: In số từ 0…9 và tính giai thừa
- Chú ý tránh các vòng lặp lồng nhau vì ảnh hưởng performance

```
for (int number = 0; number < 10; number++)
{
    Console.Write(number + " ");
}
```

```
decimal factorial = 1;
for (int i = 1; i <= n; i++)
{
    factorial *= i;
}
```

Vòng lặp foreach…

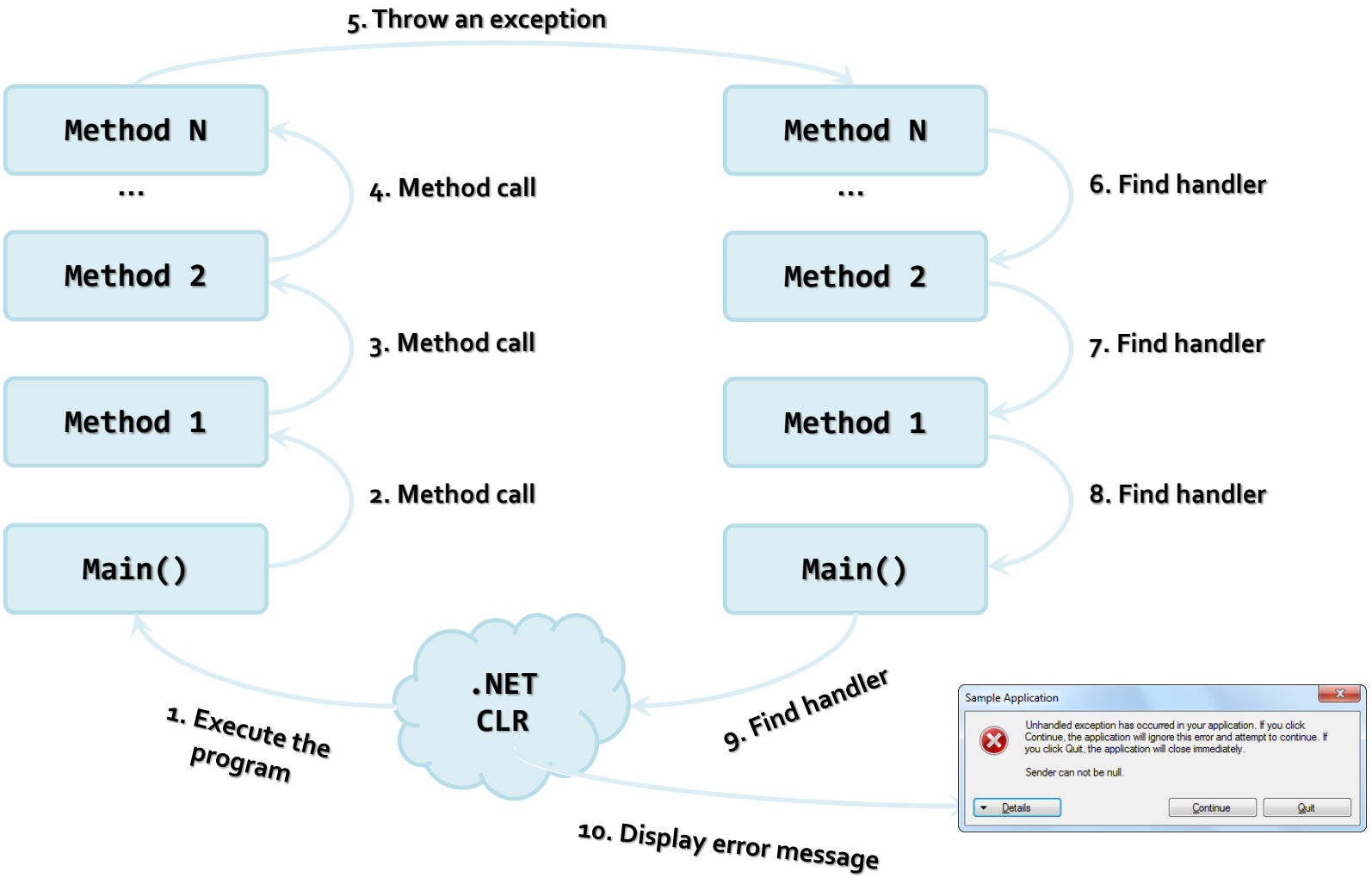
- Lặp qua các phần từ của 1 tập hợp (mảng, list). VD: lặp qua các phần tử của 1 mảng các ngày trong tuần

```
string[] days = {
  "Monday", "Tuesday", "Wednesday", "Thursday",
  "Friday", "Saturday", "Sunday" };
foreach (string day in days)
{
  Console.WriteLine(day);
}
```

Xử lý ngoại lệ (Exception)

- Sử dụng cấu trúc try…catch…finally.

```
string s = Console.ReadLine();
try
{
    Int32.Parse(s);
    Console.WriteLine(
       "You entered valid Int32 number {0}.", s);
}
catch (FormatException)
{
    Console.WriteLine("Invalid integer number!");
}
catch (OverflowException)
{
    Console.WriteLine(
       "The number is too big to fit in Int32!");
}
```


5. Throw an exception
Method N
...
Method 2
Method 1
Main()
4. Method call
3. Method call
2. Method call
Method N
...
Method 2
Method 1
Main()
6. Find handler
7. Find handler
8. Find handler
.NET CLR
1. Execute the program
9. Find handler
10. Display error message
Sample Application
Unhandled exception has occurred in your application. If you click Continue, the application will ignore this error and attempt to continue. If you click Quit, the application will close immediately.
Sender can not be null.
Details
Continue
Quit

Generic

```
public class GenericList<T>
{
  public void Add(T element) { … }
}

class GenericListExample
{
  static void Main()
  {
    // Declare a list of type int
    GenericList<int> intList =
      new GenericList<int>();

    // Declare a list of type string
    GenericList<string> stringList =
      new GenericList<string>();
  }
}
```

T là kiểu chưa xác định

T có thể được dùng trong bất kì phương thức nào của class

T được thay thế bởi kiểu int, string… khi khởi tạo

Tóm lược

- Biết cách tạo và sử dụng class, object, property, method…
- Phân biệt và sử dụng được instance type/method và static type/method
- Biết cách chuyển kiểu, sử dụng vòng lặp, xử lý exception, generic type


THANK YOU