C# and .NET Framework
Bài 3: Lập trình nâng cao trong C#

Đoàn Quang Minh
minhdqtt@gmail.com
http://www.VTPortal.net
Last update: 21. Dec 2006

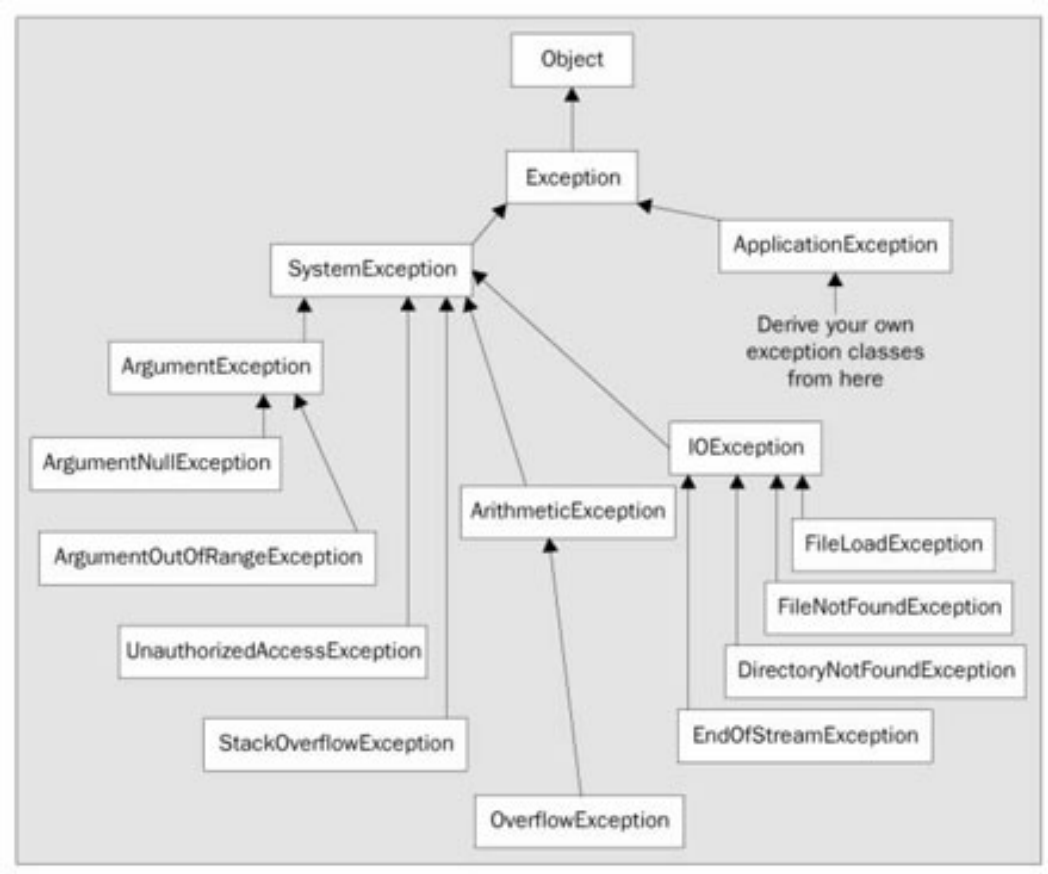
Mục lục

- Exceptions
- User-Defined Casts
- Delegates
- Events
- Generics
- Preprocessor Directive
- Unsafe code

Errors and Exception Handling

- Lỗi luôn luôn tồn tại, cho dù hệ thống được thiết kế tốt thế nào
  - Lỗi không được quyền truy cập
  - Lỗi do thiết bị hỏng (đía lỗi)
  - Lỗi do đường truyền mạng hỏng
- Khi một lỗi xuất hiện, .NET sẽ ném ra một ngoại lệ
  - Các ngoại lệ đều kế thừa từ lớp Exception
  - Tất cả các ngoại lệ cơ bản đều được cung cấp bởi .NET
  - Nếu gặp lỗi, chúng ta nên ném ra một ngoại lệ đặc biệt trong đó có mô tả thông tin rõ ràng về ngoại lệ đó
  - Nếu không tìm được lớp ngoại lệ phù hợp, có thể xây dựng lớp ngoại lệ của riêng mình

Errors and Exception Handling



Errors and Exception Handling

```
try
{
    …
}
catch (Exception e)
{
    …
}
finally
{
    …
}
```

Errors and Exception Handling

- Phần *try* thực thi các lệnh bình thường
- Phần *catch* xử lý các ngoại lệ có thể xuất hiện
  - Nếu không sử dụng ngoại lệ ném ra, có thể bỏ qua phần đối tượng đó.
  - Có thể có nhiều phần *catch* trong một khối *try catch*, khi đó mỗi phần catch xử lý một ngoại lệ khác nhau.
  - Ngoại lệ có thể được ném lại bằng từ khóa *throw*.
- Phần *finally* thực thi các lệnh kết thúc của khối lệnh.
- Ví dụ trong việc xử lý tập tin
  - Phần *try* thực thi các lệnh như mở file, đọc ghi bình thường
  - Phần *catch* xử lý lỗi.
  - Phần *finally*, nếu file được mở thì phải đóng lại

User-Defined Casts

- Toán tử ép kiểu *as*
- Chúng ta thường xuyên phải ép kiểu trong C#
  - Có 2 loại ép kiểu trong C#: implicit (an toàn tuyệt đối), explicit(có rủi ro)
  - Có thể ép kiểu từ lớp kế thừa về lớp cơ sở, song không thể làm ngược lại
  - Có thể ép kiểu qua lại giữa 2 lớp, khi đó cần khai báo hàm ép kiểu

public static implicit operator *conv-type-out* **(** *conv-type-in operand* **)**

public static explicit operator *conv-type-out* **(** *conv-type-in operand* **)**

Delegates

- delegate
  - Có những công việc không xác định lúc biên dịch, chỉ xác định lúc thực thi.
  - Các thuật toán tổng quát, ví dụ sắp xếp: không thể định nghĩa phương thức so sánh 2 đối tượng bất kỳ
  - delegate là kiểu tham chiếu, giống như class (trong C#), về ý nghĩa giống con trỏ hàm trong C++

[*attributes*] [*modifiers*] **delegate** *result-type identifier* **(**[*formal-parameters*]**);**

Delegates

```
delegate void MyDelegate(int i);

class Program
{
  public static void Main()
  {
    TakesADelegate(new MyDelegate(DelegateFunction));
  }

  public static void TakesADelegate(MyDelegate SomeFunction)
  {
    SomeFunction(21);
  }

  public static void DelegateFunction(int i)
  {
    System.Console.WriteLine("Called by delegate with number: {0}.", i);
  }
}
```

Events

- Sự kiện
  - Được sử dụng để báo hiệu một điều gì đó xảy ra.
  - Trong Windows, có rất nhiều sự kiện.
  - Trong C#, event là một dạng đặc biệt của delegate

  [*attributes*] [*modifiers*] **event** *type declarator*; [*attributes*] [*modifiers*] **event** *type member-name* {*accessor-declarations*};
- Phát sinh sự kiện
  - Định nghĩa tham số sự kiện, đặt tên là *EventName*EventArgs, kế thừa từ System.EventArgs.
  - Định nghĩa một delegates cho sự kiện, đặt tên là *EventName*EventHandler.
  - Phát sinh sự kiện
    - Khai báo sự kiện
    - Khai báo một phương thức On*EventName* để phát sinh sự kiện

Events

- Xử lý sự kiện
  - Nếu một component phát sinh một sự kiện, có thể bắt và xử lý sự kiện đó.
  - Để hander sự kiện trong Windows Form hoặc trong Web Form:
    - Khai báo component (ví dụ button)
    - Khai báo hàm xử lý sự kiện
    - Gắn hàm vào sự kiện

Generics

- Generics
  - Cho phép class, struct, interface, method sử dụng kiểu dữ liệu mà nó lưu trữ như là tham số đầu vào.
  - Khái niệm giống như template của C++.
  - Generics xuất hiện nhằm mục đích xử lí chính xác kiểu của dữ liệu. Ví dụ, với stack, nếu không có generics thì dữ liệu coi như các object, nên đòi hỏi phải ép kiểu khi xử lí, điều này có thể gây lỗi run-time.
- Tạo và sử dụng Generics
  - Khai báo giống như template trong C++: dùng cặp dấu < >
  - Sử dụng: phải chỉ định chính xác kiểu dữ liệu

Generics

- Ví dụ sử dụng Generics
  - Khai báo:

```
public class Stack<ItemType>
{
    private ItemType[] items = new ItemType[100];
    public void Push(ItemType data) {…}
    public ItemType Pop() {…}
}
```

  - Sử dụng

```
Stack<int> s = new Stack<int>();
s.Push(3);
int x = s.Pop();
```

Preprocessor Directives

- #define and #undef
- #if, #elif, #else, and #endif
- #warning and #error
- #region and #endregion
- #line

Memory Management

- C# tự động quản lý bộ nhớ nhờ vào bộ thu gom rác
  - Bộ nhớ ảo trong Windows
  - Stack và heap
- Có 2 loại kiểu dữ liệu trong C#
  - Value Data Types:
    - Dữ liệu chứa tại nơi nó được cấp phát vùng nhớ
    - Các kiểu số, bool, char, date, các cấu trúc, các kiểu liệt kê
  - Reference Data Types:
    - Chứa một con trỏ trỏ tới nơi cất giữ dữ liệu
    - Bao gồm kiểu string, mảng, class, delegate

Unsafe Code

- C# tự quản lý bộ nhớ, tuy nhiên có những lúc chúng ta cần sử dụng con trỏ.
  - Dùng từ khóa unsafe tại vùng lệnh muốn sử dụng con trỏ
  - Phải có tham số biên dịch unsafe khi dịch chương trình
- Con trỏ:
  - Các khai báo và sử dụng tương tự C++

Tài liệu tham khảo

- Professional C#, Second Edition
- http://www.asp.net
- http://www.microsoft.com/net/default.mspx
- http://www.codeproject.com
- Địa chỉ download tài liệu http://www.thanglong.edu.vn/giang-day/tab.aspx
- Diễn đàn C# & .NET http://www.thanglong.edu.vn/forum/cmd/0/category/hoc-tap-nghien-cuu/dot-net/tab.aspx