



# EXCEPTION HANDLING



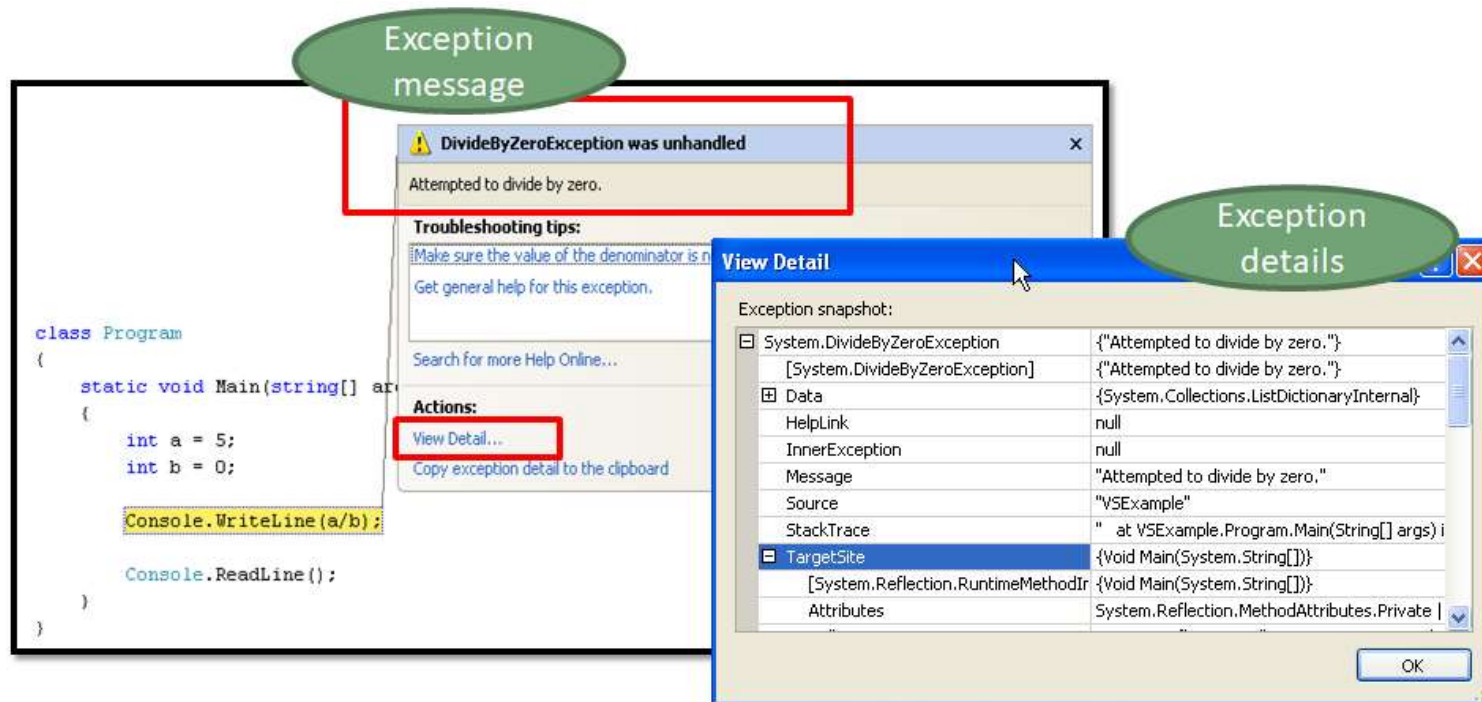


# EXCEPTION HANDLING

- Errors, Bugs, Exceptions.
- Catching Exception: try – catch – finally.
- Exception Propagation.

# ERRORS, BUGS, EXCEPTIONS

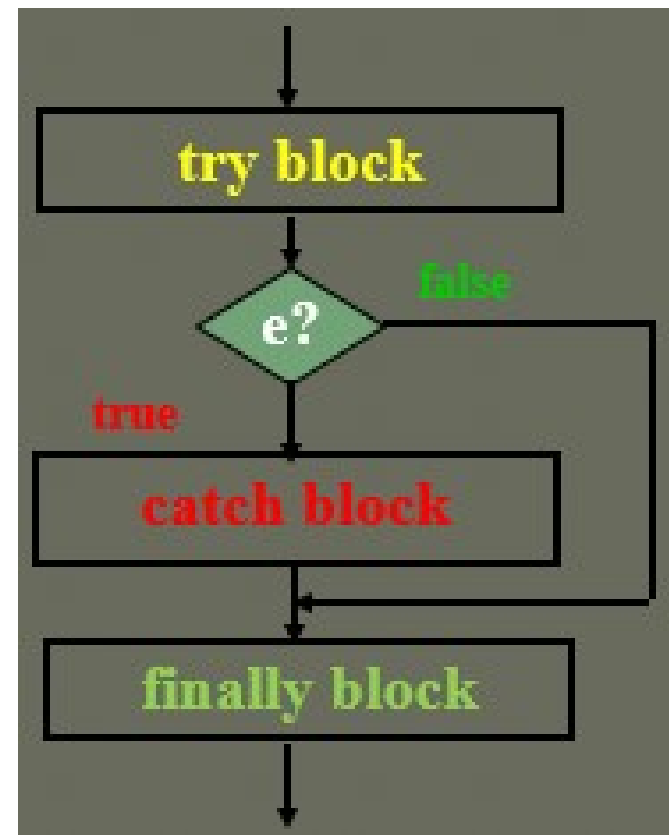
- Bugs: an error on the part of the programmer:
  - For example, making calls on a NULL pointer.
- User errors: typically caused by the individual running the application:
  - For example, an end user who enters a malformed string into a text box.
- Exceptions: run time error and beyond the control of the program.



# CATCHING EXCEPTIONS

*try – catch – finally*

```
try
{
    //protected code
}
catch
{
    //error handling code
}
finally
{
    //optional cleanup code
}
```



# CATCHING EXCEPTIONS

```
class Program
{
    static void Main(string[] args)
    {
        try
        {
            int a = 5;
            int b = 0;

            Console.WriteLine(a / b);
        }
        catch (DivideByZeroException dex) // Specific exception
        {
            Console.WriteLine("Divide By Zero Exception: " + dex.Message);
        }
        catch (Exception ex) // General exception
        {
            Console.WriteLine("General Exception: " + ex.Message);
        }
        finally
        {
            Console.WriteLine();
            Console.WriteLine("This is always called, whatever it has exception!");
        }

        Console.ReadLine();
    }
}
```

File: H:\G:\FUDemo\CS\Sharp.NET\FSE\Example\FSEExample\bin\Debug\\*

Divide By Zero Exception: Attempted to divide by zero.

This is always called, whatever it has exception!

-



# CATCHING EXCEPTIONS

```
namespace ExceptionTutorial
```

```
{
```

```
    0 references
```

```
    class HelloException
```

```
    {
```

```
        0 references
```

```
        public static void Main(string[] args)
```

```
        {
```

```
            Console.WriteLine("Three");
```

```
            int value = 10 / 2;
```

```
            Console.WriteLine("Two");
```

```
            value = 10 / 1;
```

```
            Console.WriteLine("One");
```

```
            int d = 0;
```

```
            value = 10 / d;
```

```
            Console.WriteLine("Let's go!");
```

```
            Console.Read();
```

```
        }
```

```
    }
```

```
}
```

Lỗi làm ngừng  
chương trình  
xảy ra tại đây

```
namespace ExceptionTutorial
```

```
{
```

```
    0 references
```

```
    class HelloCatchException
```

```
    {
```

```
        0 references
```

```
        public static void Main(string[] args)
```

```
        {
```

```
            Console.WriteLine("Three");
```

```
            int value = 10 / 2;
```

```
            Console.WriteLine("Two");
```

```
            value = 10 / 1;
```

```
            Console.WriteLine("One");
```

```
            int d = 0;
```

```
            try
```

```
            {
```

```
                value = 10 / d;
```

```
                Console.WriteLine("Value = " + value);
```

```
            } catch (DivideByZeroException e)
```

```
            {
```

```
                Console.WriteLine("Error: " + e.Message);
```

```
                Console.WriteLine("Ignore...");
```

```
            }
```

```
            Console.WriteLine("Let's go!");
```

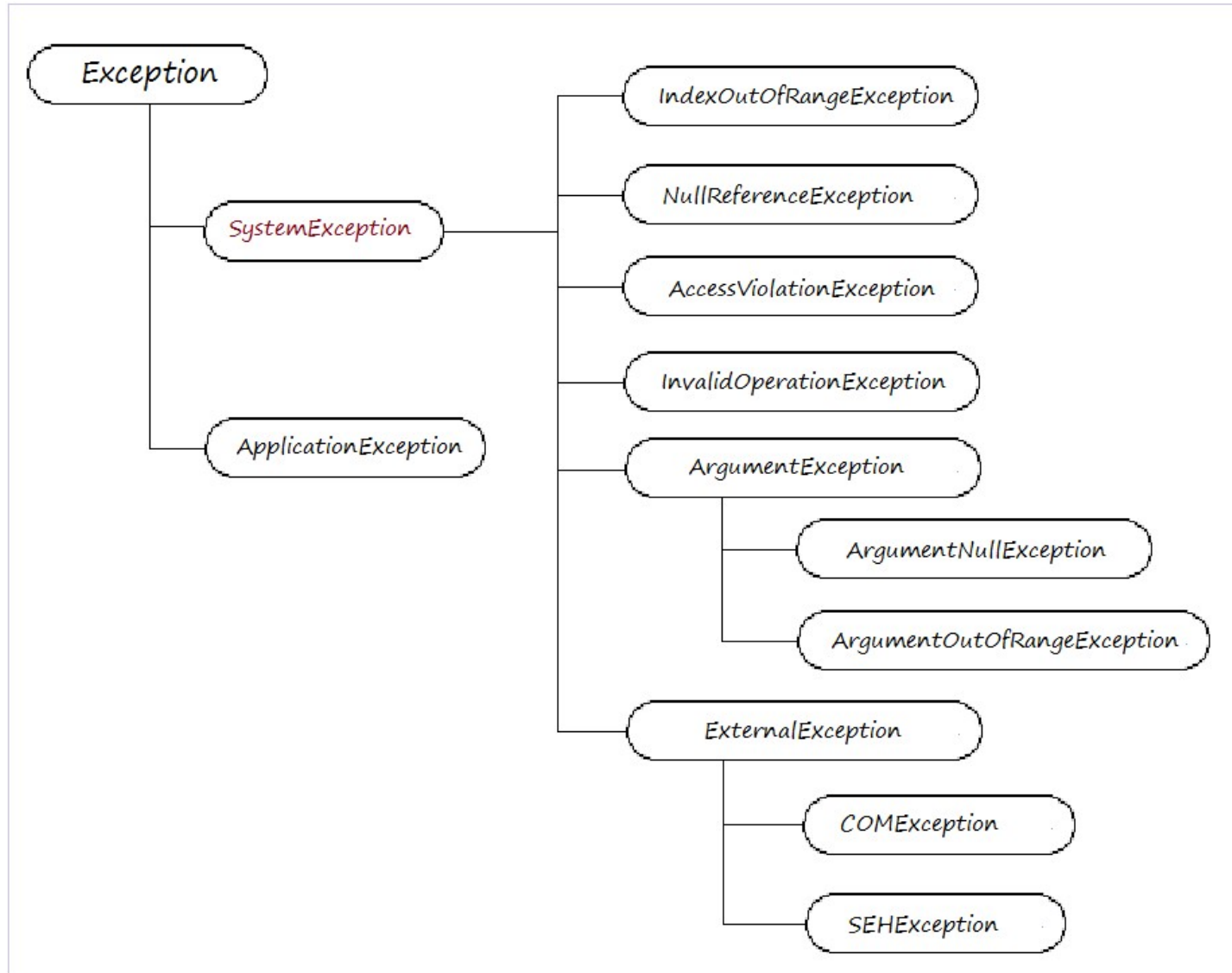
```
            Console.Read();
```

```
        }
```

```
}
```



# EXCEPTION C#





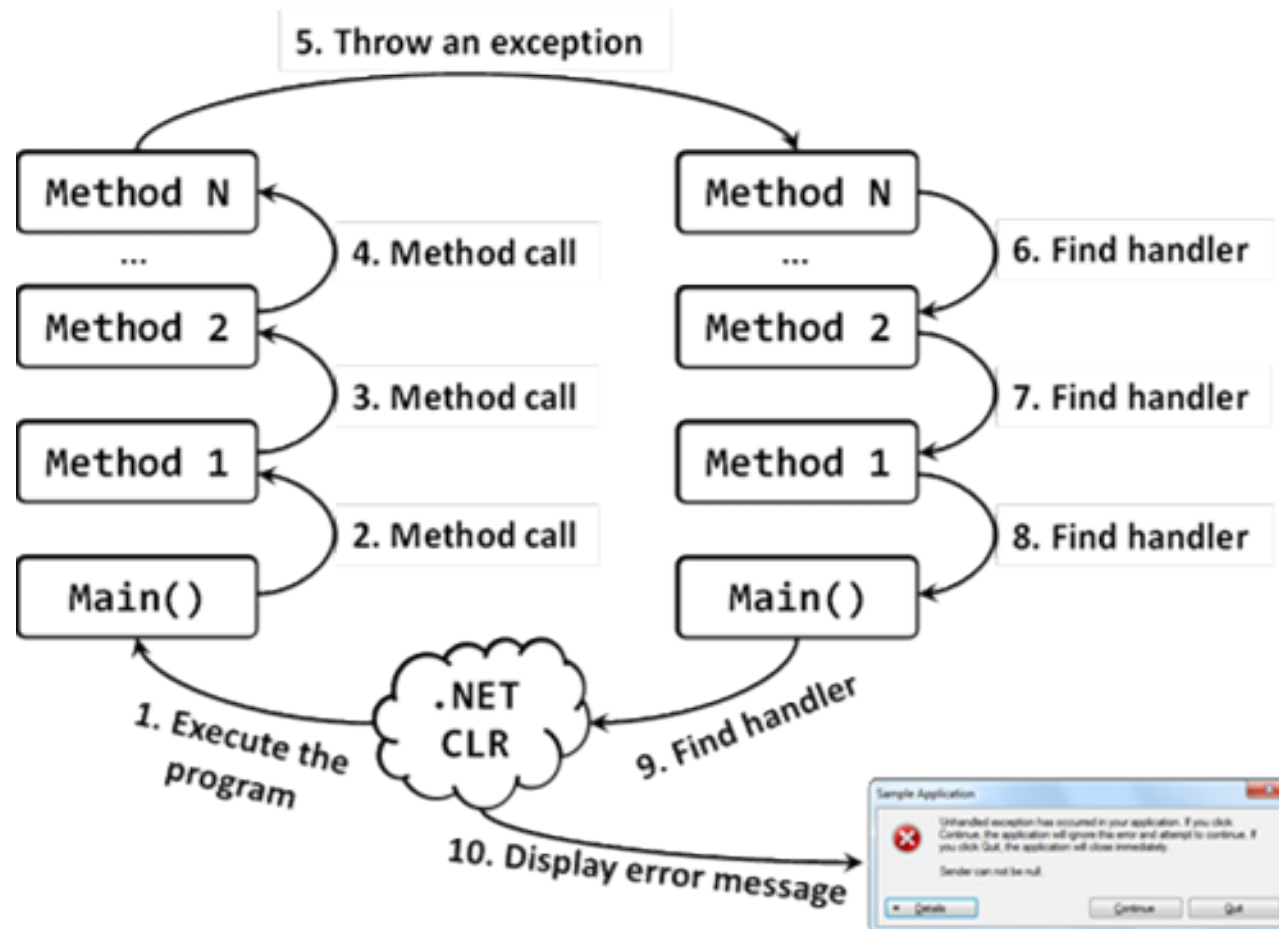
# EXCEPTION C#

Kiểu ngoại lệ	Mô tả
Exception	Lớp cơ bản của mọi ngoại lệ.
SystemException	Lớp cơ bản của mọi ngoại lệ phát ra tại thời điểm chạy của chương trình.
IndexOutOfRangeException	Được ném ra tại thời điểm chạy khi truy cập vào một phần tử của mảng với chỉ số không đúng.
NullReferenceException	Ném ra tại thời điểm chạy khi một đối tượng <b>null</b> được tham chiếu.
AccessViolationException	Ném ra tại thời điểm chạy khi tham chiếu vào vùng bộ nhớ không hợp lệ.
InvalidOperationException	Ném ra bởi phương thức khi ở trạng thái không hợp lệ.
ArgumentException	Lớp cơ bản cho các ngoại lệ liên quan tới đối số (Argument).
ArgumentNullException	Lớp này là con của <b>ArgumentException</b> , nó được ném ra bởi phương thức mà không cho phép thông số <b>null</b> truyền vào.
ArgumentOutOfRangeException	Lớp này là con của <b>ArgumentException</b> , nó được ném ra bởi phương thức khi một đối số không thuộc phạm vi cho phép truyền vào nó.
ExternalException	Lớp cơ bản cho các ngoại lệ xảy ra hoặc đến từ môi trường bên ngoài.
COMException	Lớp này mở rộng từ <b>ExternalException</b> , ngoại lệ đóng gói thông tin COM.
SEHException	Lớp này mở rộng từ <b>ExternalException</b> , nó tóm lược các ngoại lệ từ Win32.



# EXCEPTION PROPAGATION

- Method call and exception handling process:



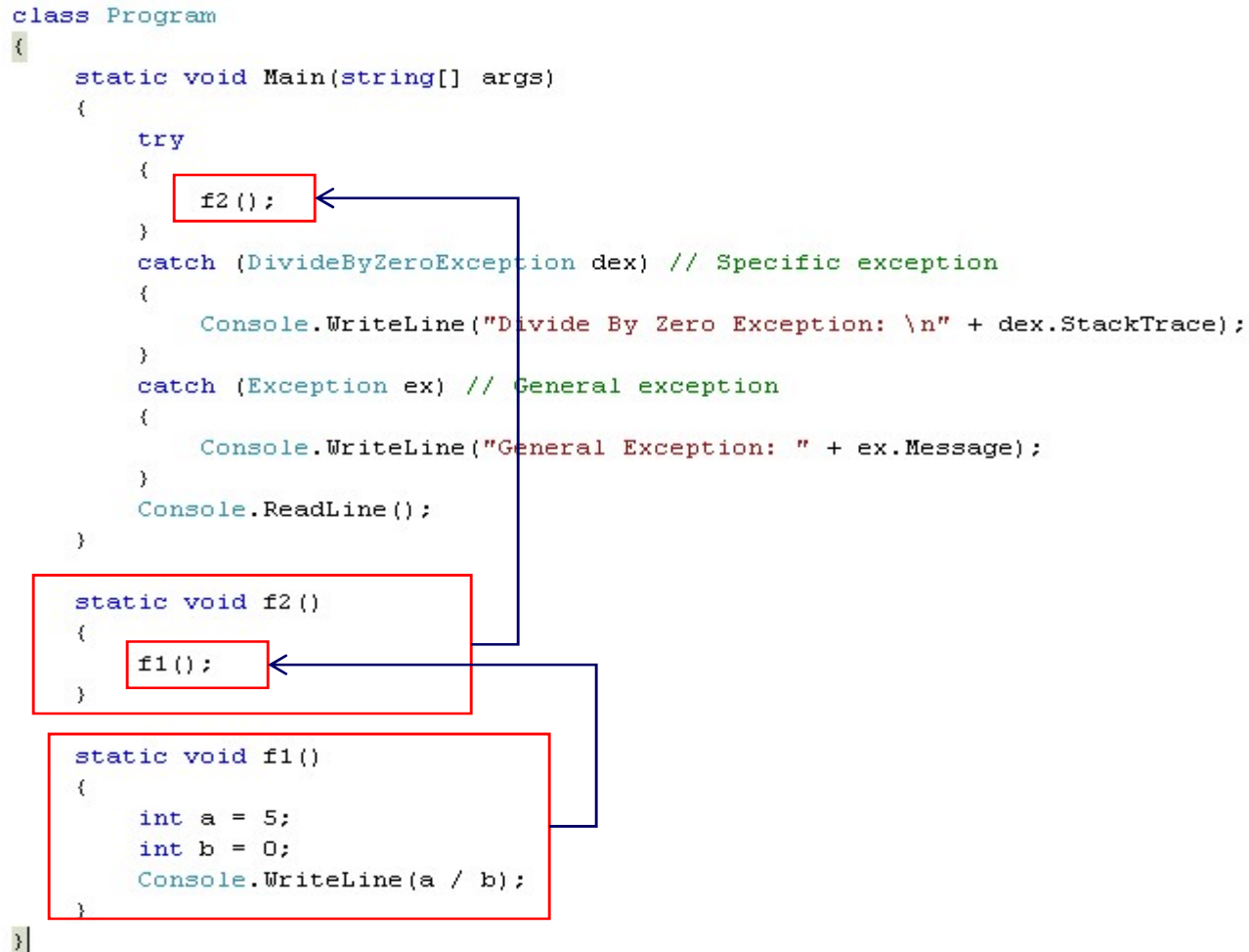
# EXCEPTION PROPAGATION

- Method call and exception handling process:

```
class Program
{
    static void Main(string[] args)
    {
        try
        {
            f2();
        }
        catch (DivideByZeroException dex) // Specific exception
        {
            Console.WriteLine("Divide By Zero Exception: \n" + dex.StackTrace);
        }
        catch (Exception ex) // General exception
        {
            Console.WriteLine("General Exception: " + ex.Message);
        }
        Console.ReadLine();
    }

    static void f2()
    {
        f1();
    }

    static void f1()
    {
        int a = 5;
        int b = 0;
        Console.WriteLine(a / b);
    }
}
```



The diagram illustrates the flow of an exception during a method call. It shows three methods: `Main`, `f2`, and `f1`. `Main` calls `f2`, which in turn calls `f1`. `f1` contains a division by zero operation (`a / b` where `b` is 0), which throws a `DivideByZeroException`. The exception propagates back up the call stack: first to `f2`, and then to `Main`, where it is caught by the `catch (DivideByZeroException dex)` block. Red boxes highlight the call sites and the exception handling blocks, while blue arrows trace the path of the exception from the point of occurrence in `f1` to its final handling in `Main`.

Thank You !

