# C# Programming

Zheng-Liang Lu

Department of Computer Science & Information Engineering
National Taiwan University

Online Course

```
class Lecture8
{
            "Exceptions and Exception Handling"
}

// Keywords:
try, catch, finally, throw
```

## Introduction[2]

- Errors in the program at run time are propagated through the program by using a mechanism called exceptions. [1]
  - For example, open a missing file.
- When an error occurs in one method, the method creates an exception object and hands it off to the runtime system.
- This is called throwing an exception.
- The runtime system searches the call stack for a method that contains a block of code that can handle the exception, called exception handler.

[1]Note that the exception should be a force majeure.

[2]See `https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/exceptions/`.

## The Handling Blocks: try-catch-finally

- Exception handling uses the try, catch, and finally keywords to try actions that may not succeed.
- First we use a try block around the statements that might throw exceptions.
- The catch keyword is used to define an exception handler.
- Do not catch an exception unless you can handle it and leave the application in a known state.
- If you catch **Exception**, re-throw it using the throw keyword at the end of the catch block.
- Code in a finally block is executed even if an exception is thrown.
  - We often use a finally block to release resources.

```
...
    static void Main(string[] args)
    {
        Console.WriteLine("Enter an integer?");
        try
        {
            int x = int.Parse(Console.ReadLine());
        }
        catch (FormatException e)
        {
            Console.WriteLine("Not an integer.");
        }
        catch (Exception e)
        {
            Console.WriteLine("Unknown exception.");
        }
        finally
        {
            Console.WriteLine("Cleanup is done.");
        }
        Console.WriteLine("End of program.");
    }
...
```

## Another Example: Safe Division

```
...
    static double SafeDivision(double x, double y)
    {
        if (y == 0) throw new DivideByZeroException();
        return x / y;
    }

    static void Main(string[] args)
    {
        double a = 1, b = 0, result = 0;
        try
        {
            result = SafeDivision(a, b);
            Console.WriteLine("{0} divided by {1} = {2}"
                                , a, b, result);
        }
        catch (DivideByZeroException e)
        {
            Console.WriteLine("Attempted divide by zero.");
        }
    }
...
```

## Throwing Exceptions

- We sometimes disallow the behaviors from users.
- Exceptions can be explicitly generated by a program via using the throw keyword.

```
class Circle
{
    double Radius { get; set; }

    public Circle(double radius)
    {
        if (radius <= 0)
                throw new Exception("Not a good radius?");
        Radius = radius;
    }
}
```

## Customized Exceptions

- We create our own exceptions by deriving from **Exception**.

```
class NegativeRadiusException : Exception
{
    public NegativeRadiusException(double radius)
                        : base("The radius is invalid: " + radius) { }

}
```

```
class Circle
{
    double Radius { get; set; }

    public Circle(double radius)
    {
        if (radius <= 0)
            throw new NegativeRadiusException(radius);
        Radius = radius;
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        new Circle(-10); // This will produce an exception.
    }
}
```

Fin.