C#: Using and Handling Exceptions

Eric Fisher
INTERACTIVE COURSE AUTHOR, PLURALSIGHT

@EricJamesFisher

Exceptions

Are designed to handle unexpected or exceptional situations that occur when a program is running.

What we'll cover

Throwing Exceptions

Using Try / Catch Statements

Using Try / Catch / Finally Statements

Throwing Exceptions

The throw keyword followed by an exception object will throw that exception

Example.cs

```
public void ThrowException(string[] a)
{
    throw new ArgumentException("Human friendly exception message");
}
```

Throw will stop execution by throwing the provided exception

Any type of exception can be thrown. (ArgumentException, etc)

Do not use exceptions to change the flow of a program as part of ordinary execution! Exceptions should only be used to report and or handle error conditions.

Try-Catch Statement

Try-Catch Statements consist of a try block and a catch block

Example.cs

```
public void GetReport()
{
    try
    {
        ReadFileFromHardDrive();
    }
    catch(Exception ex)
    {
        ReportError(ex);
        throw;
    }
}
```

Try Block

Try blocks are wrapped around code that can potentially cause problems

Example.cs

```
public void GetReport()
{
    try
    {
        ReadFileFromHardDrive();
    }
    catch(Exception ex)
    {
        ReportError(ex);
        throw;
    }
}
```

First our program will try to run the code in the try block.

If Try Succeeds

If the `try` block runs without throwing an exception it'll skip the catch block

Example.cs

```
public void GetReport()
{
    try
    {
        ReadFileFromHardDrive();
    }
    catch(Exception ex)
    {
        ReportError(ex);
        throw;
    }
}
```

The catch block only runs when try throws an exception

If Try Fails

If the code in the `try` block throws an exception we'll run the catch block

Example.cs

```
public void GetReport()
{
    try
    {
        ReadFileFromHardDrive();
    }
    catch(Exception ex)
    {
        ReportError(ex);
        throw;
    }
}
```

The catch block's argument is set to the exception was thrown in the try block

Catch Block Arguments

The exception thrown by the try block will be the argument of the catch block

Example.cs

```
public void GetReport()
{
    try
    {
        ReadFileFromHardDrive();
    }
    catch(Exception ex)
    {
        ReportError(ex);
        throw;
    }
}
```

When throwing from a catch block you do not need to specify the exception to be thrown

Handling Different Exception Types

Catch blocks can also be used to catch only specific exception types

Example.cs

```
try
{
    ReadFileFromHardDrive();
}
catch(FileNotFoundException ex)
{
    FixFileNotFound();
}
catch(Exception ex)
{
    ReportError(ex);
    throw;
}
```

This catch will only be executed if there is a FileNotFoundException

This catch will executed for any exception that isn't a FileNotFoundException

Finally Block

A `finally` block executes code regardless of if an exception is caught or not

Example.cs

```
try
{
    ReadFileFromHardDrive();
}
catch(Exception ex)
{
    ReportError(ex);
}
finally()
{
    DisposeOfFileStream();
}
```

Finally will run regardless of if an exception occurs or not and are generally used to cleanup resources to prevent side effects when an exception occurs

Tip: Don't Rethrow Catch Argument

When rethrowing an error do not throw the captured exception

Example.cs

```
catch(Exception ex)
{
    ReportError(ex);
    throw;
}
```

When throwing in a catch block, throw will automatically rethrow the exception that caused the catch to execute.

```
catch(Exception ex)
{
    ReportError(ex);
    throw ex;
}
```

Rethrowing the exception argument can cause you to lose the stack trace of that exception. (making debugging harder)

Tip: Catch Should Never Just Throw

Catching and then just immediately throwing serves no purpose

Example.cs

```
catch(Exception ex)
{
    ReportError(ex);
    throw;
}

catch(Exception ex)
{
    throw;
}
```

When you catch an exception you should do something, even if it's just logging that it happened

If you catch then immediately rethrow an exception the results are the same as if you never caught the exception in the first place. (except it has a slightly higher resource cost)

Tip: Don't Leave Catch Blocks Empty

Empty catch blocks “hide” exceptions making them difficult to fix

Example.cs

```
try
{
    ReadFileFromHardDrive();
}
catch(Exception ex)
{

}
```

Having an empty catch block effectively hides exceptions, this prevents you from seeing problems and makes debugging them harder.

Summary

Exceptions are used to handle unexpected or exceptional situations.

`throw` throws whatever exception follows it. (or the caught exception when called inside a catch block)

`Try` / `Catch` will attempt to run code, if code in the `try` block throws an exception then the code in the `catch` block will run

`Finally` will run regardless of if an exception occurs or not in a `try` block and is generally used to clean up allocated resources