Exceptions

What Is an Exception?

- An exception is an indication of an error or exceptional condition
- Exceptions propagate
- .NET provides many built-in exceptions:
  - **Exception**
  - **SystemException**
  - **ApplicationException**
  - **NullReferenceException**
  - **FileNotFoundException**
  - **SerializationException**

Handling Exception by Using a Try/Catch Block

- Use try/catch blocks to handle exceptions
- Use one or more catch blocks to catch different types of exceptions
- Let’s practice this

```
try
{
}
catch (NullReferenceException ex)
{
    // Catch all NullReferenceException exceptions.
}
catch (Exception ex)
{
    // Catch all other exceptions.
}
```

Using a Finally Block

Use a finally block to run code whether or not an exception has occurred

```
try
{
}
catch (NullReferenceException ex)
{
    // Catch all NullReferenceException exceptions.
}
catch (Exception ex)
{
    // Catch all other exceptions.
}
finally
{
   // Code that always runs.
}
```

Practicing exception catches

- Please see PracticingExceptionCatches folder.

Throwing Exceptions

- Use the **throw** keyword to throw a new exception

```
var ex =
   new NullReferenceException("The 'Name' parameter is null.");
throw ex;
```

- Use the **throw** keyword to rethrow an existing exception

```
try
{
}
catch (NullReferenceException ex)
{
}
catch (Exception ex)
{
    ...
    throw;
}
```

Creating Custom Exceptions

To create a custom exception type:

1. Inherit from the **System.Exception** class
2. Implement three standard constructors:
   - base()
   - base(string message)
   - base(string message, Exception inner)
3. Add additional members if required

Throwing and Catching Custom Exceptions

- Use the **throw** keyword to throw a custom exception

```
throw new LoyaltyCardNotFoundException();
```

- Use a try/catch block to catch the exception

```
try
{
   // Perform the operation that could cause the exception.
}
catch(LoyaltyCardNotFoundException ex)
{
   // Use the exception variable, ex, to get more information.
}
```