Турсунов Баходурхон

#23. .NET Exceptions/Errors and error handling. Exceptions, Throw Exception. Built-In Exceptions. Try…Catch…Finally… block. Custom Exceptions. Compiler-generated exceptions.

**1. Как создать пользовательские исключения в C#?**

Пользовательские исключения создаются путем определения нового класса, который наследуется от класса Exception или его производных.

**using** System;  
  
// Создаем пользовательское исключение MyCustomException  
**public** **class** **MyCustomException** : **Exception**  
{  
 // Конструктор с возможностью передачи сообщения об ошибке  
 **public** **MyCustomException**(**string** message) : **base**(message)  
 {  
 }  
  
 // Дополнительные конструкторы или свойства могут быть добавлены по необходимости  
}  
  
**class** **Program**  
{  
 **static** **void** **Main**()  
 {  
 **try**  
 {  
 // Генерируем пользовательское исключение с сообщением  
 **throw** **new** MyCustomException("Это мое исключение!");  
 }  
 **catch** (MyCustomException ex)  
 {  
 // Перехватываем и обрабатываем пользовательское исключение  
 Console.WriteLine($"Поймано исключение: {ex.Message}");  
 }  
 **catch** (Exception ex)  
 {  
 // Перехватываем другие типы исключений  
 Console.WriteLine($"Поймано другое исключение: {ex.Message}");  
 }  
 }  
}

**2. Каково назначение блока 'try...catch...finally' в обработке исключений? Опишите каждую часть этого блока.**

Блок try catch finally используется для обработки исключений. Этот блок позволяет писать код, который может вызывать исключения, и предоставляет механизм для их обработки и выполнения определенных действий даже в случае возникновения исключения.

Блок try. В этом блоке помещается код, который может вызывать исключение. Если внутри блока try происходит исключение, управление передается блоку catch для обработки. В блоке catch указывается тип исключения, который мы хотим обработать, и код для обработки этого исключения.

Блок finally. Этот блок содержит код, который будет выполнен в любом случае, независимо от того, произошло исключение или нет. Этот блок обычно используется для выполнения завершающих действий, таких как закрытие файлов, освобождение ресурсов и т. д. Этот блок не является обязательным, но он предоставляет удобный способ гарантированного выполнения кода, даже если произошло исключение.

**3. Как можно получить дополнительную информацию о типе возникшего исключения при обработке исключений в C#?**

Это можно сделать, используя объект исключения и его свойства. Когда исключение перехватывается в блоке catch, информация о самом исключении становится доступной через переменную, которая указывается в объявлении catch.

**try**  
{  
 // Код, который может вызвать исключение  
 **throw** **new** InvalidOperationException("Это исключение не допустимо!");  
}  
**catch** (InvalidOperationException ex)  
{  
 // Обработка исключения типа InvalidOperationException  
 Console.WriteLine($"Поймано исключение типа {ex.GetType().Name}: {ex.Message}");  
}  
**catch** (Exception ex)  
{  
 // Обработка других типов исключений  
 Console.WriteLine($"Поймано другое исключение: {ex.GetType().Name} - {ex.Message}");  
}  
**finally**  
{  
 // Код, который выполнится независимо от того, произошло исключение или нет  
}

4. Какими свойствами обладают все исключения в C#, и какую информацию об ошибке они предоставляют?

В C# исключения являются объектами, производными от класса System.Exception. Этот базовый класс определяет набор свойств и методов, которые присутствуют во всех исключениях. Некоторые из основных свойств, которыми обладают все исключения, включают:

* Message, который содержит текстовое описание ошибки.
* StackTrace, который содержит строковое представление стека вызовов в момент возникновения исключения. Это позволяет отследить, как программа достигла места, где произошло исключение.
* InnerException, если исключение возникло в контексте другого исключения, то это свойство содержит ссылку на внутреннее исключение
* GetType() – метод, который возвращает тип объекта исключения.

5. Можно ли иметь несколько блоков catch с разными классами исключений, соединенных в цепочку? Как они оцениваются и выполняются?

Да, мы можем иметь несколько блоков catch с разными классами исключений, и они будут соединены в цепочку. Когда возникает исключение, система исполняет блоки catch в порядке, в котором они следуют в коде, и выбирает первый подходящий блок для обработки исключения. Это позволяет обрабатывать различные типы исключений в специфическом порядке.

На пример:

try  
{  
 // Код, который может вызвать исключение  
 **throw** **new** InvalidOperationException("Недопустимая операция!");  
}  
catch (ArgumentException ex)  
{  
 Console.WriteLine($"Обработка ArgumentException: {ex.Message}");  
}  
catch (InvalidOperationException ex)  
{  
 Console.WriteLine($"Обработка InvalidOperationException: {ex.Message}");  
}  
catch (Exception ex)  
{  
 Console.WriteLine($"Общая обработка исключения: {ex.Message}");  
}

**6. Как блок 'finally' помогает освободить ресурсы в C#? Приведите пример использования блока 'finally' для закрытия файла.**

Блок finally используется для выполнения кода, который должен быть гарантированно выполнен, независимо от того, возникло исключение или нет. Это очень полезно для освобождения ресурсов, таких как файлы, сетевые подключения, базы данных и т. д.

Пример использования блока finally для закрытия файла.

**using** System;  
**using** System.IO;  
  
**class** **Program**  
{  
 **static** **void** **Main**()  
 {  
 FileStream fileStream = null;  
  
 **try**  
 {  
 // Открываем файл для чтения или записи  
 fileStream = **new** FileStream("example.txt", FileMode.OpenOrCreate);  
  
 // Делаем что-то с файлом, например, читаем или пишем в него  
 // ...  
  
 // Здесь может возникнуть исключение  
 // throw new Exception("Some exception");  
 }  
 **catch** (Exception ex)  
 {  
 Console.WriteLine($"Произошло исключение: {ex.Message}");  
 }  
 **finally**  
 {  
 // В блоке finally освобождаем ресурсы, например, закрываем файл  
 **if** (fileStream != null)  
 {  
 **try**  
 {  
 fileStream.Close();  
 }  
 **catch** (Exception ex)  
 {  
 Console.WriteLine($"Ошибка при закрытии файла: {ex.Message}");  
 }  
 }  
 }  
 }  
}

Practice:

Напишите программу на C#, которая демонстрирует следующее:

1. Использование блока try...catch для обработки определенного исключения. Вы можете создать простую операцию деления, которая пытается разделить на ноль (например, 10 / 0) и поймать исключение DivideByZeroException. В блоке catch выведите сообщение об ошибке на консоль.

2. Используйте пользовательское исключение, создав свой собственный класс исключений (например, MyCustomException), производный от Exception. В своей программе намеренно вызовите это пользовательское исключение с помощью ключевого слова throw.

3. Используйте блок finally, чтобы гарантировать, что определенные ресурсы (например, файл, соединение с базой данных или сетевое соединение) будут должным образом закрыты или освобождены независимо от того, было ли выброшено исключение...