1. Расскажите как генерируется исключение.

Исключения могут генерироваться либо исполняющей средой, либо пользовательским кодом.

Программная среда обнаруживает ошибку выполнения программы, такую как деление на ноль, выход за пределы массива и т. д. Встреченная ошибка приводит к созданию объекта исключения, который содержит информацию о типе ошибки и контексте ее возникновения. Программная среда бросает созданный объект исключения.

Разработчики могут явно бросить исключение с использованием ключевого слова `throw`.

2. Расскажите методику обработки исключений.

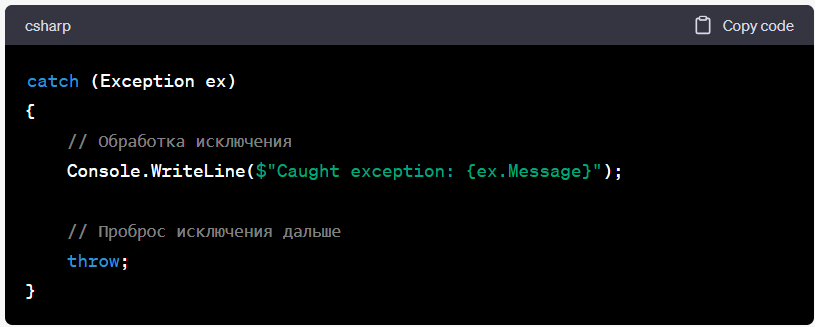
1. \*\*Блок `try`:\*\* В этом блоке помещается код, который может вызвать исключение. Если в блоке `try` происходит исключение, выполнение кода в этом блоке останавливается, и управление передается соответствующему блоку `catch`.

2. \*\*Блок `catch`:\*\* Этот блок содержит код, который выполняется, если в блоке `try` произошло исключение. Блок catch имеет доступ к объекту Exception, который содержит информацию о происшедшей ошибке. Блок catch применяется либо для обработки ошибки, либо для повторной генерации исключения

- `ExceptionType`: Это тип исключения, который вы хотите перехватить. Может быть конкретным типом исключения, например, `DivideByZeroException`, или более общим типом, таким как `Exception`. **Exception** является базовым классом для всех исключений в языке программирования C#. Вся иерархия исключений строится на основе этого класса.

3. \*\*Блок `finally`:\*\* Этот блок содержит код, который выполняется всегда, независимо от того, произошло исключение или нет. Блок `finally` является необязательным.

4. \*\*Проброс исключения:\*\* Если блок `catch` не может обработать исключение, вы можете пробросить его дальше, добавив `throw;` внутри блока `catch`. Это позволяет другим участкам кода обработать исключение.



3. Какое ключевое слово служит для обозначения блока кода, в котором можно генерировать исключение?

В C# ключевое слово throw используется для генерации исключения в блоке кода. Оно позволяет явным образом бросить (сгенерировать) исключение в процессе выполнения программы.

4. Какие ключевые слова используются для обработки и генерации исключений? Расскажите об механизме обработке исключения?

Механизм:

1) Исключение не произошло

1.1. try выполняем до конца

1.2. catch пропускаем

1.3. finally выполняем

2) Исключение произошло

2.1. выполнение try прекращается (все что идет за возникшим исключение игнорируется)

2.2 ищем блок catch на соответствие по типу исключения

2.2.1. если нет catch

2.2.1.1 разматывает стек, локальные объекты, выходят из области видимости

2.2.1.2 снова генерируется исключение в точке вызова метода

2.2.1.3. если блока не найдено, то сообщение - необработанное исключении, дальнейшее выполнение программы останавливается

2.2.2 сatch найден

2.2.2.1 Передается управление ближайшему **catch-обработчику, совместимому с** типом выброшенного исключения

2.2.2.2. объект-исключения передается, если это предусмотрено, обработчику в качестве параметра.

2.3. переходим/ищем finally

2.3.1. если нет finnaly

2.3.1.1.выполнение программы продолжается начиная с позиции, след. за последним обработчиком данного блока try

5. Что будет, если в программе нет предложения catch, способного обработать исключение?

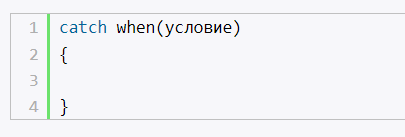
1. \*\*Прекращение выполнения текущего метода:\*\* Исключение будет передано в вызывающий метод, и так далее вверх по стеку вызовов. Если и там нет блока `catch`, обрабатывающего данное исключение, оно будет передано дальше, вплоть до точки входа в программу (обычно метод `Main`).

2. \*\*Завершение программы:\*\* Если исключение доходит до метода `Main` и здесь также нет подходящего блока `catch`, программа завершится. В этом случае обычно будет выведено сообщение о необработанном исключении, и программа закроется.

3. \*\*Вывод стека вызовов (Call Stack):\*\* Если исключение не обработано, часто выводится информация о стеке вызовов, что может помочь в определении места и причины ошибки.

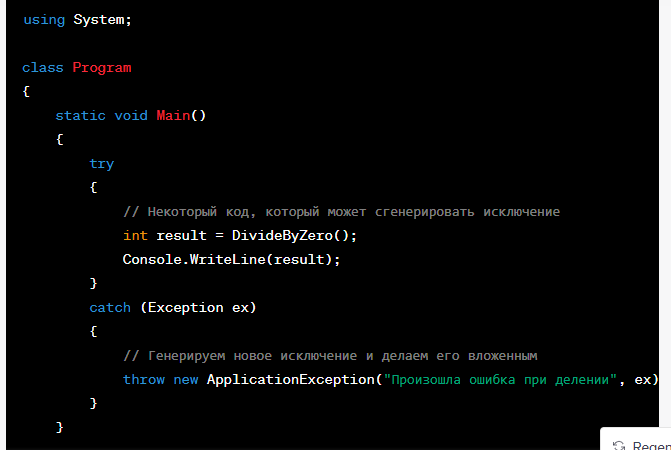
6. Что такое фильтры исключения? Приведите пример

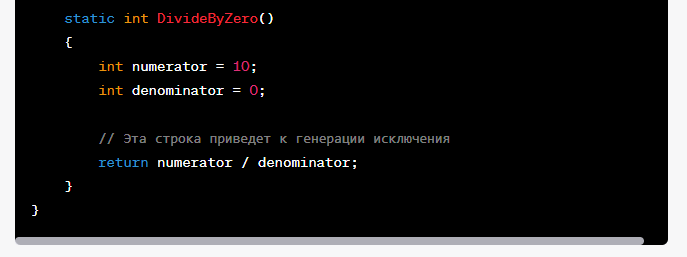
Фильтры исключений позволяют обрабатывать исключения в зависимости от определенных условий. Для их применения после выражения catch идет выражение **when**, после которого в скобках указывается условие:



7. Могут ли исключения быть вложенными?

Да. Вложенные исключения происходят, когда одно исключение вызывает другое. Когда в блоке `catch` обработчика исключения генерируется новое исключение, это новое исключение может быть вложенным в предыдущее.





В приведенном выше примере исключение `DivideByZeroException`, которое возникает при делении на ноль, будет перехвачено блоком `catch`, и новое исключение `ApplicationException` будет сгенерировано с предыдущим исключением (`ex`), делая его вложенным.

При обработке вложенных исключений важно учитывать, что вы можете использовать свойство `InnerException`, чтобы получить доступ к предыдущему исключению в стеке вызовов.

Одни конструкции try-catch могут содержать другие. Если исключение возникает во вложенной конструкции try-catch, то программа сначала ищет во вложенной конструкции блок catch, который обрабатывает нужный тип исключения. Если во вложенной конструкции try-catch такой блок catch не найден, то программа начинает искать аналогичный блок catch во внешей конструкии try-catch.

8. Какой синтаксис нужно использовать в C# для отлова любого возможного исключения?

В C#, чтобы отловить любое возможное исключение, вы можете использовать базовый тип `Exception`. В блоке `catch` указывается тип исключения, который вы хотите обработать. Если вы используете `Exception`, это будет отлавливать все исключения, потому что `Exception` является базовым классом для всех исключений в C#. Либо catch без параметров.

9. Чем следует руководствоваться при размещении обработчиков исключения?

1. Предпочтительнее явно указывать тип исключения, который вы хотите обработать, вместо использования обработчика для всех исключений (`Exception`). Это позволяет более точно определить, какие ошибки вы хотите обрабатывать, и предотвращает скрытие проблем в коде.

2. Не стоит ловить исключения, если вы не знаете, как обрабатывать конкретный тип. Если вы не уверены, что делать с исключением, дайте ему всплыть вверх по стеку вызовов.

3. В обработчиках исключений часто следует использовать логгирование (`Logging`). Это помогает отслеживать ошибки в продакшене, обеспечивает информацию для отладки и аудита.

4. Используйте `catch` только там, где действительно нужно обработать ошибку. Иногда лучше дать приложению упасть, чем продолжить его выполнение в нестабильном состоянии.

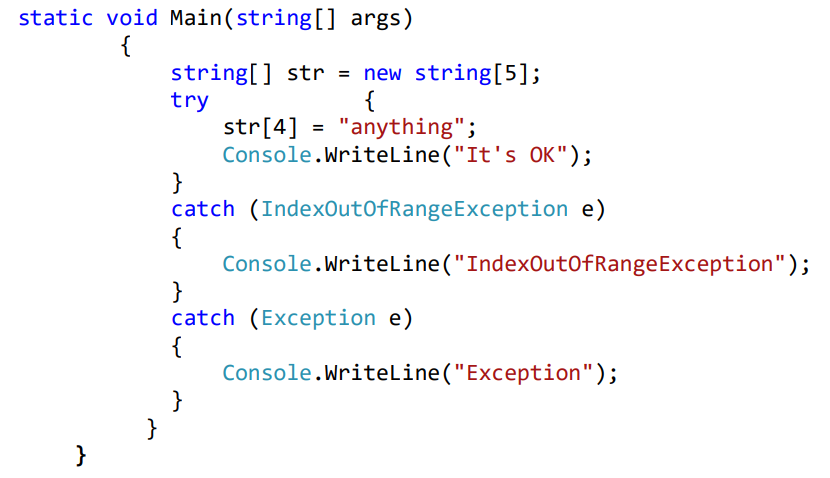
5. Если у вас есть ресурсы, которые требуют освобождения, используйте блок `finally` для выполнения таких действий независимо от того, произошло исключение или нет.

6. Используйте иерархию исключений для разделения различных типов ошибок и их обработки на разных уровнях приложения.

Свойства:

* try могут быть вложенные
* более специфичные исключения обрабатываются первыми
* Свои классы исключений должны наследоваться от System.Exсeption или System.ApplicationException
* Генерация исключений в finally нежелательно - код восстановления или очистки будет выполнен не полностью
* процедура обработки исключений медленная

10.Что будет выведено на консоль в результате выполнения фрагмента листинга?

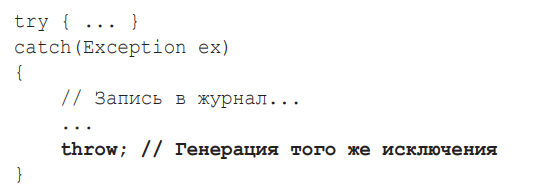


В результате выполнения данного фрагмента листинга будет выведено следующее: It's OK

Поскольку операция `str[4] = "anything";` не вызывает исключение (индекс 4 находится в пределах допустимого диапазона для массива `str`), выполнится блок `try`, и на консоль будет выведено "It's OK". Блоки `catch` не будут вызваны в данном случае.

11.Как повторно сгенерировать то же самое исключение в блоке обработчике catch?

Исключение можно перехватывать и генерировать повторно:



Повторная генерация в подобной манере дает возможность зарегистрировать в журнале информацию об ошибке, не подавляя ее. Она позволяет также отказаться от обработки исключения, если обстоятельства сложились не так, как ожидалось.

12.Какие методы содержаться в классе Exception? Где и как их можно использовать?

InnerException: хранит информацию об исключении, которое послужило причиной текущего исключения

Message: хранит сообщение об исключении, текст ошибки

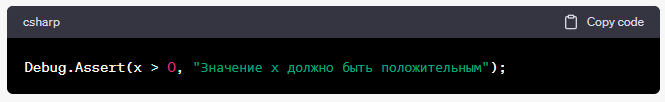
Source: хранит имя объекта или сборки, которое вызвало исключение

StackTrace: возвращает строковое представление стека вызывов, которые привели к возникновению исключения

TargetSite: возвращает метод, в котором и было вызвано исключение

Assert

В C#, в отличие от некоторых других языков программирования, таких как C++, нет прямых эквивалентов макросов `assert`. Вместо этого в C# используется стандартная библиотека тестирования и встроенный класс `Debug` для поддержки подобной функциональности.



`Debug.Assert` - это метод в классе `Debug`, предоставляемом .NET Framework, который используется для вставки проверок утверждений (assertions) в код во время разработки.

Если условие, переданное методу `Assert`, оценивается как `false`, то будет сгенерировано исключение `Debug.AssertionFailedException`, и выполнение программы остановится.

Debug и Debugger

1. \*\*Debug:\*\*

- Вывод отладочной информации (`Debug.WriteLine`).

- Проверки утверждений (`Debug.Assert`).

- Прерывания выполнения (`Debug.Fail`).

2. \*\*Debugger:\*\*

- Принудительное прерывание (`Debugger.Break()`).

- Проверка наличия отладчика (`Debugger.IsAttached`).

- Настройка отладчика (`Debugger.Launch()`).

- Атрибуты отладки, управление отображением значений.