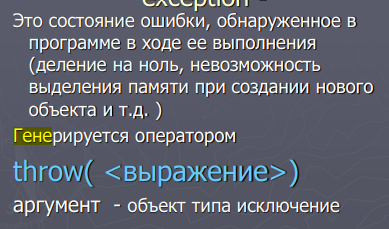
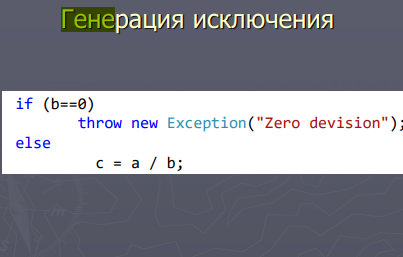
**ВОПРОСЫ К ЛАБОРАТОРНОЙ РАБОТЕ №7:**

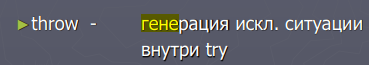
1. Расскажите как генерируется исключение.

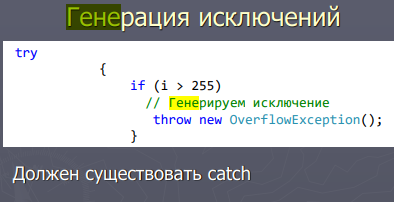
Исключительная ситуация – состояние ошибки, обнаруженное в программе в ходе ее выполнения (/0, невозможно выделение памяти…)

if (b==0)  
 throw new Exception(“Zero div”);









2. Расскажите методику обработки исключений.

При использовании блока **try...catch..finally** вначале выполняются все инструкции в блоке **try**. Если в этом блоке не возникло исключений, то после его выполнения начинает выполняться блок **finally**. И затем конструкция try..catch..finally завершает свою работу.

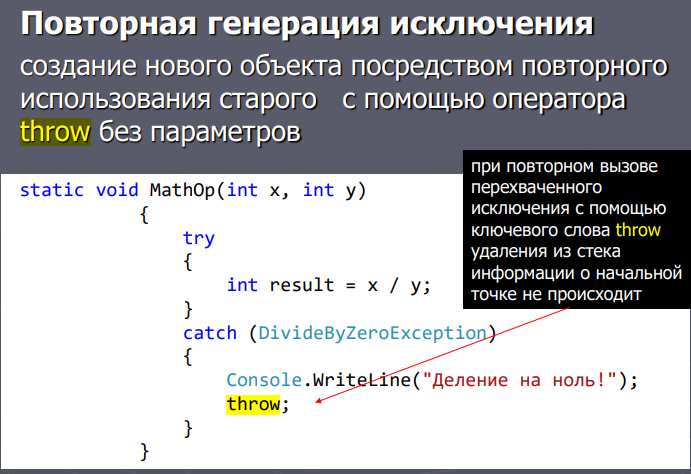
Если же в блоке try вдруг возникает исключение, то обычный порядок выполнения останавливается, и среда CLR начинает искать блок **catch**, который может обработать данное исключение. Если нужный блок catch найден, то он выполняется, и после его завершения выполняется блок finally.

Если нужный блок catch не найден, то при возникновении исключения программа аварийно завершает свое выполнение.

try – контролируемый блок  
throw – генерация искл. ситуации внутри try  
catch – обработчик исключений  
finally – код, очищающий ресурсы и др. д-вия (выполнение всегда, 1 на 1 try)

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

**Throw**

****

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

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?

Если подходящих обработчиков **catch** не существует, то выполнение программы переходит к следующему блоку **try**. Если до конца программы не найдены соответствующие обработчики **catch**, то программа завершает свое выполнение с ошибкой исключения.

Код в блоке **finally** выполняется, даже если создано исключение. Используйте блок **finally**, чтобы освободить ресурсы, например закрыть потоки и файлы, которые были открыты внутри блока **try**.

6. Что такое фильтры исключения? Приведите пример.

Фильтр искл. – позв. указать доп. условия, при кот. исп. обработчик исключения  
Эти условия bool, перед ними **WHEN:**

catch (Exception ex) when  
 (ex.GetType() != typeof(System.FormatException))  
 … //обработка исключений  
 //кроме того, кот. называется FormatException

7. Могут ли исключения быть вложенными?

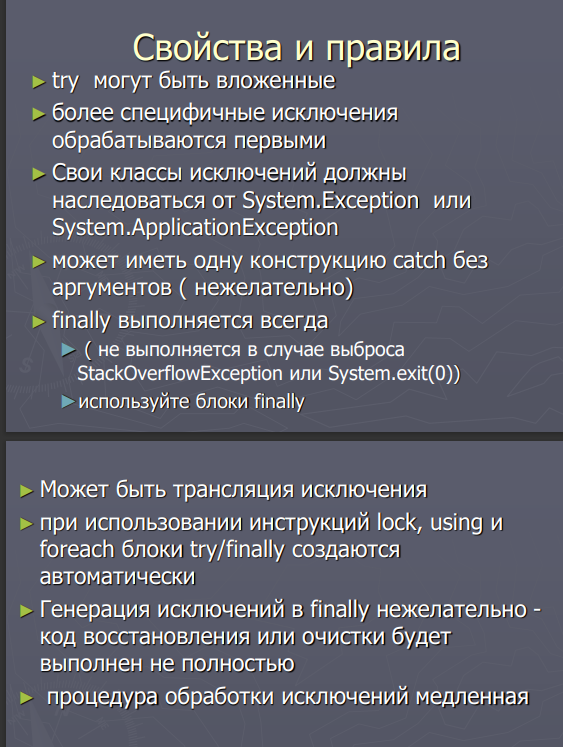
**try** может

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

Блок **catch**, отлавливающий исключения типа System.Exception. Также можно просто использовать catch{}, не указывая тип исключения.

9. Чем следует руководствоваться при размещении обработчиков исключения?

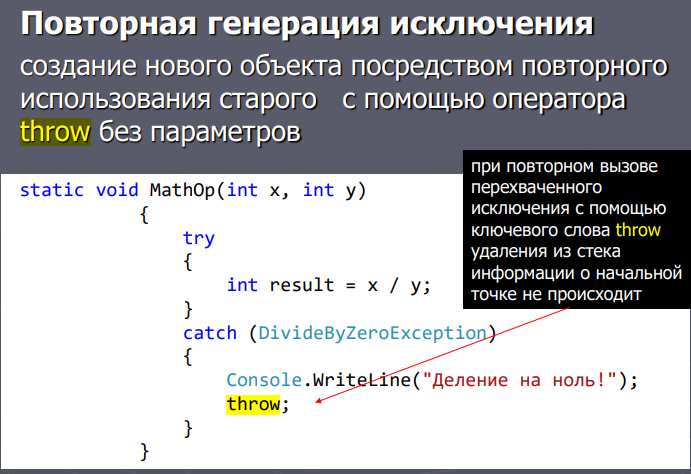
\* сначала более специфичное исключение  
\* далее наследуется от System.Exception или System.ApplicationException  
\* мб один catch без аргумент (нежелательно)  
\* finally вып всегда  
\* при lock, using, foreach блоки try/finally созд. авто-  
\* генерация исключения в finally нежелательно



10. It’s okay

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

try  
 int res = x / y  
catch (DivideByZeroException)  
 Console.WriteLine(“Деление на ноль!”);  
 throw; //при повт. вызове исключения с пом. throw  
. //инфа о нач. точке ост. в стеке



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

**Exception** — это базовый класс, от которого наследуются исключения. Например, иерархия класса InvalidCastException имеет следующий вид:

*Object*

*Exception*

*SystemException*

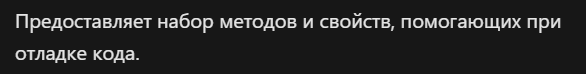
*InvalidCastException*

Все определяемые на уровне пользователя и системы исключения в конечном итоге всегда наследуются от базового класса System.Exception, который, в свою очередь, наследуется от класса System.Object.

Exception.GetBaseException  
При переопределении в производном классе возвращает исключение Exception, которое является первопричиной одного или нескольких последующих исключений.  
  
GetBaseException Используйте метод, если требуется найти первопричину исключения, но не требуется информация об исключениях, которые могли возникнуть между текущим исключением и первым исключением.  
  
Exception.GetObjectData(SerializationInfo, StreamingContext)  
При переопределении в производном классе задает объект SerializationInfo со сведениями об исключении.

|  |  |
| --- | --- |
| **Свойство** | **Описание** |
| *Data* | Это свойство, доступное только для чтения, позволяет извлекать коллекцию пар "ключ/значение" (представленную объектом, реализующим интерфейс IDictionary), которая предоставляет дополнительную, определяемую программистом, информацию об исключении. По умолчанию эта коллекция является пустой. |
| *HelpLink* | Это свойство позволяет получать или устанавливать URL-адрес, по которому доступен справочный файл или веб-сайт с детальным описанием ошибки. |
| *InnerException* | Это свойство, доступное только для чтения, может применяться для получения информации о предыдущем исключении или исключениях, которые послужили причиной возникновения текущего исключения. Запись предыдущих исключений осуществляется путем их передачи конструктору самого последнего исключения. |
| *Message* | Это свойство, доступное только для чтения, возвращает текстовое описание соответствующей ошибки. Само сообщение об ошибке задается в передаваемом конструктору параметре. |
| *Source* | Это свойство позволяет получать или устанавливать имя сборки или объекта, который привел к выдаче исключения. |
| *StackTrace* | Это свойство, доступное только для чтения, содержит строку с описанием последовательности вызовов, которая привела к возникновению исключения. Как нетрудно догадаться, это свойство очень полезно во время отладки или для сохранения информации об ошибке во внешнем журнале ошибок. |
| *TargetSite* | Это свойство, доступное только для чтения, возвращает объект MethodBase с описанием многочисленных деталей метода, который привел к выдаче исключения (вызов вместе с ним ToString() позволяет идентифицировать этот метод по имени) |

Debug



Asert, fail

Debugger

