Занятие 4, ч1. Перегрузка, обработка исключений, управление ресурсами в .Net

Тренер: Алексей Дышлевой

Основные вопросы

- Перевантаження операцій. Типи й формати перевантажених операцій. Операнди та їх порядок. Операції, що допускають перевантаження. Перевантаження арифметичних операцій, операцій порівняння, операцій перетворення, булевих операцій. операцій присвоювання.
- Обробка виключень. Генерація виключень. Необроблені виключення. Оператори try, catch, finally. Повторна генерація виключень. Виключення у блоці finally. Виключення у фіналізаторах. Виключення у конструкторах. Обмежені області виконання. Критичні фіналі затори та SafeHandle. Користувацькі класи виключень. Забезпечення відкочування змін.
- Життєвий цикл об’єкта. Керовані ресурси в .Net. Клас GC. Деструктор.
- Управління ресурсами. Знищення об’єктів. Фіналізатори. Детерміноване знищення. Інтерфейс IDisposable
- Тип System.Object. Интерфейс IClonable.
- Еквівалентність типів. Інтерфейс IComparable

Перегрузка операций

- Перегруженные операции определяются как общедоступные статические методы в классах, для которых они предназначены
- struct Hour
- {
- public static Hour operator +(Hour lhs, Hour rhs)
- {
- return new Hour(lhs.value + rhs.value);
- }
- }

Перегрузка операций

- Перегруженная операция является методом класса, который принимает множество параметров и возвращает значение. При этом операнды передаются в виде параметров, а возвращаемое значение – результат операции.
- Возможна перегрузка одного оператора в классе несколько раз с разными аргументами.
- В заголовке операции обязательно должен быть хотя бы один аргумент включающего его типа.
- Все методы должны быть static. Перегруженные операции не могут быть полиморфными и использовать модификаторы virtual, abstract, override или sealed.

Вызов перегруженных операций

- Hour a, Hour b;
- a + b;
- operator+ (a, b);

Категории операций перегрузки

- Унарные операции («!», «++», «--», «+» и «-»). При перегрузке указывается один параметр, который должен быть того же типа, что и класс, который определяет оператор.
- Бинарные операции («*», «/», «+», «-» и «%»). При перегрузке этих операций необходимо указать два параметра, по крайней мере один из которых должен быть того же типа, что и класс, определяющий операцию.
- Операции преобразования. Эти операции можно использовать для преобразования данных одного типа в другой. При перегрузке этих операций, необходимо указать один параметр, содержащий данные, которые нужно конвертировать. Эти данные могут быть любого допустимого типа.

Возможности перегрузки операций

Операции	Возможность перегрузки
+, -, !, ~, ++, --, true, false	Унарные операции, которые могут быть перегружены.
+, -, *, /, %, &, |, ^, <<, >>	Бинарные операции, которые могут быть перегружены.
==, !=, <, >, <=, >=	Операции сравнения, которые могут быть перегружены.
&&, ||	Условные логические операции не могут быть перегружены, но они оцениваются с помощью & и |, которые могут быть перегружены.
[ ]	Операция индексирования массива не может быть перегружена, но можно определить индексаторы, которые могут бть перегружены.
()	Операция приведения не может быть перегружена, но можно определить новые операции преобразования.
+=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=	Операции присваивания не могут быть перегружены, но +=, например, оценивается с помощью операции «+», которая может быть перегружена.
=, ., ?:, ->, new, is, sizeof, typeof	Эти операции не могут быть перегружены.

Класс Exception (конструкторы)

Имя	Описание
Exception()	Инициализирует новый экземпляр класса Exception.
Exception(String)	Выполняет инициализацию нового экземпляра класса Exception, используя указанное сообщение об ошибке.
Exception(SerializationInfo, StreamingContext)	Инициализирует новый экземпляр класса Exception с сериализованными данными.
Exception(String, Exception)	Инициализирует новый экземпляр класса Exception указанным сообщением об ошибке и ссылкой на внутреннее исключение, которое стало причиной данного исключения.

Ограничения перегрузки операций

- Нельзя изменить приоритет или ассоциативность операции. Приоритет и ассоциативность основаны на символе операции (например, «+») и не зависят от типа (например, int), для которого используется символ операции. Следовательно, выражение a + b * c всегда то же, что и выражение a + (b * c), независимо от типов a, b и c.
- Нельзя изменить множественность (число операндов) операции. Например, символ умножения «*» определяет бинарную операцию. При объявлении в типе операции «*» она должна быть бинарной операцией. Аналогично, операция «++» является унарной операцией, и если она объявляется в типе, она должна быть унарной операцией.
- Нельзя придумать новые символы операции. Например, нельзя создать новый символ операции, такой как «**», для возведения в степень. Если необходимо выполнить операцию, для которой не существует оператора, необходимо создать метод.
- Нельзя изменить смысл операций по отношению ко встроенным типам. Например, выражение 1 + 2 имеет предопределенное значения, и нельзя его переопределить. Когда в типе определяется операция, по крайней мере один из операндов для этой операции должен быть содержащего типа.
- Кроме того, операторы сравнения необходимо реализовать в парах. Например, если перегружается операция «>», необходимо также перегрузить операцию «<». Если перегружается операция «==», необходимо также перегрузить операцию «!=».
- Если в классе определяются операции «==» и «!=», также необходмо переопределить методы GetHashCode и Equals, унаследованные от System.Object (или System.ValueType при создании структуры). Метод Equals должен обладать точно таким же поведение, как оператор «==». (Следует определить одно через другое.) Метод GetHashCode используется другими классами в Microsoft .NET Framework (например, при использовании объекта в качестве ключа в хэш-таблице).

Рекомендации

- Не изменять операнды.
- Определять симметрические операции.
- Определять только имеющие смысл операции.

Задача

- Перегрузить операторы для сравнения двух отрезков и проверки в условных операторах. Предусмотреть возможности операций с разными типами (int, double, string, отрезок) и в разной последовательности (первый или второй операнд).

Операции преобразования

- Синтаксис объявления операции пользовательского преобразования аналогичен объявлению перегруженной операции.
- Операция преобразования должна быть public и static.
- Имя операции преобразования либо implicit (если она реализует расширяющее преобразование) или explicit (если он реализует сужающее преобразование).
- public static implicit operator int (Hour from)
- {
- return from.value;
- }

Операции преобразования

- Сужающее преобразование
- public Hour(int hr)
- {
- this.value = hr % 24;
- }
- public static explicit operator Hour (int from)
- {
- return new Hour(from);
- }

Операции преобразования

- Операции преобразования предоставляют альтернативный способ решения проблемы обеспечения симметрических операций.
- public Hour(int hr)
- {
- this.value = hr % 24;
- }
- public static Hour operator +(Hour lhs, Hour rhs)
- {
- return new Hour(lhs.value + rhs.value);
- }
- public static implicit operator Hour (int from)
- {
- return new Hour (from);
- }

Задача

- Изменить предыдущий пример таким образом, чтобы использовать меньшее число перегруженных операторов с помощью оператора преобразования

Обработка исключений

- Исключение является показателем ошибки или исключительного состояния приложения. Метод может сгенерировать исключение, когда обнаруживает, что случилось что-то неожиданное.
- Как только исключение сгенерировано, CLR начинает процесс итеративной раскрутки стека приложений. При этом она очищает все объекты, локальные по отношению к каждому фрейму стека.
- Если в некотором фрейме присутствует обработчик исключений, созданный специально для этого исключения, то CLR вызывает его.
- Если такой обработчик не найден, то будет сгенерировано «необработанное исключение» и работа приложения будет прервана.
- В .NET Framework исключения основаны на классе Exception, в котором содержится информация об исключении. Когда метод генерирует исключение, он создает объект Exception и может наполнить его информацией о причине ошибки. Этот объект передается обрабатывающему исключение коду, который может его использовать, чтобы определить лучший способ справиться с исключением.

Обработка исключений

- try
- {
- [Try block.]
- }
- catch ([catch specification 1])
- {
- [Catch block 1.]
- }
- ...
- catch ([catch specification n])
- {
- [Catch block n.]
- }

Обработка исключений

- Тип Exception часто используется для того, чтобы поймать все исключения, которые не были обработаны иным образом. В следующем примере, если код в блоке try является причиной исключения DivideByZeroException, управление передается соответствующему блоку catch. Если возникает какой-либо другой тип исключения, работает код в блоке catch для типа Exception.

```
try
{
    // Try block.
}
catch (DivideByZeroException ex)
{
    // Catch block, can access DivideByZeroException exception in ex.
}
catch (Exception ex)
{
    // Catch block, can access exception in ex.
}
```

Обработка исключений

```
try
{
    // Outer try block.
    ...
    try
    {
        // Nested try block
    }
    catch (FileNotFoundException ex)
    {
        // Catch block for nested try block
    }
    ...
    // Outer try block continued
}
catch (DivideByZeroException ex)
{
    // Catch block, can access DivideByZeroException exception in ex.
}
catch (Exception ex)
{
    // Catch block, can access exception in ex.
}
```

Свойства исключений (класс Exception)

Свойство	Описание
Message	Предоставляет подробные сведения о причине возникновения исключения. Свойство Message выводится на том языке, который указан в свойстве Thread.CurrentUICulture для потока, который создает исключение.
Source	Содержит строку, которая указывает на объект или приложение, которое вызвало ошибку. Если свойство не задано, возвращается имя сборки, в которой возникло исключение.
StackTrace	Содержит трассировку стека, которую можно использовать для определения места возникновения ошибки. Трассировка стека включает имя файла источника и, при наличии отладочной информации, номер программной строки.
TargetSite	Свойство является строкой, содержащей имя метода, сгенерировавшего исключение. Если метод, выбросивший исключение, недоступен и трассировка стека не является указателем NULL, свойство TargetSite извлекает метод из трассировки стека. Если трассировка стека является нулевой ссылкой, то TargetSite возвращает нулевую ссылку.
InnerException	Свойство может использоваться для создания и сохранения серии исключений во время обработки исключений. Это свойство можно использовать для создания нового исключения, содержащего ранее перехваченные исключения. Исходное исключение может быть захвачено вторым исключением в свойстве InnerException, что позволяет коду, обрабатывающему второе исключение, проверять дополнительные данные. Например, существует метод, выполняющий чтение файла и форматирование данных, который пытается выполнить чтение из файла, но при этом генерируется исключение FileException. Данный метод перехватывает FileException и создает исключение BadFormatException. В этом случае исключение FileException может быть сохранено в свойстве InnerException исключения BadFormatException.
HelpLink	Свойство может содержать URL к файлу справки, предоставляющему подробные сведения о причине возникновения исключения.
Data	Свойство является объектом типа IDictionary и может содержать произвольные данные в парах "ключ-значение", которые можно использовать для хранения дополнительной информации об ошибке.

Блок finally

- Некоторые методы могут содержать крайне необходимый код, который всегда должен быть выполнен, даже если возникает необработанное исключение.
- В него помещается код, который должен быть выполнен при завершении блока try/catch независимо от любых возникших исключений, обработанных или необработанных. При этом, если исключение перехватывается и обрабатывается, обработчик исключений в блоке catch будет работать раньше блока finally. Блок finally можно также добавить к блоку try, не имеющему блоков catch. В этом случае, все исключения являются необработанными, при этом блок finally будет работать всегда.
- try
- { [Try block.]
- }
- catch ([catch specification 1])
- { [Catch block 1.]
- }
- ...
- catch ([catch specification n])
- { [Catch block n.]
- }
- finally
- { [Finally block.]
- }

Передача управления с использованием блока finally

- Выполняется блок **try**.
- Если сгенерировано исключение:
  - Если есть соответствующий исключению блок **catch**:
    - Выполняется блок **catch**, который соответствует исключению.
    - Выполняется блок finally.
  - Если есть соответствующий исключению блок catch, и этот блок catch сам является причиной исключения:
    - Выполняется блок catch, который соответствует оригиналу исключения.
    - Выполняется блок finally.
    - Исключение, вызванное обработчиком **catch** распространяется на любой внешний блок try/catch, или вызов метода, если такого блока не существует.
  - Если нет соответствующего исключению блока catch:
    - Выполняется блок finally.
    - Исключение распространяется на любой внешний блок try/catch, или вызов метода, если такого блока не существует.
- Если исключение не сгенерировано, выполняется блок finally.

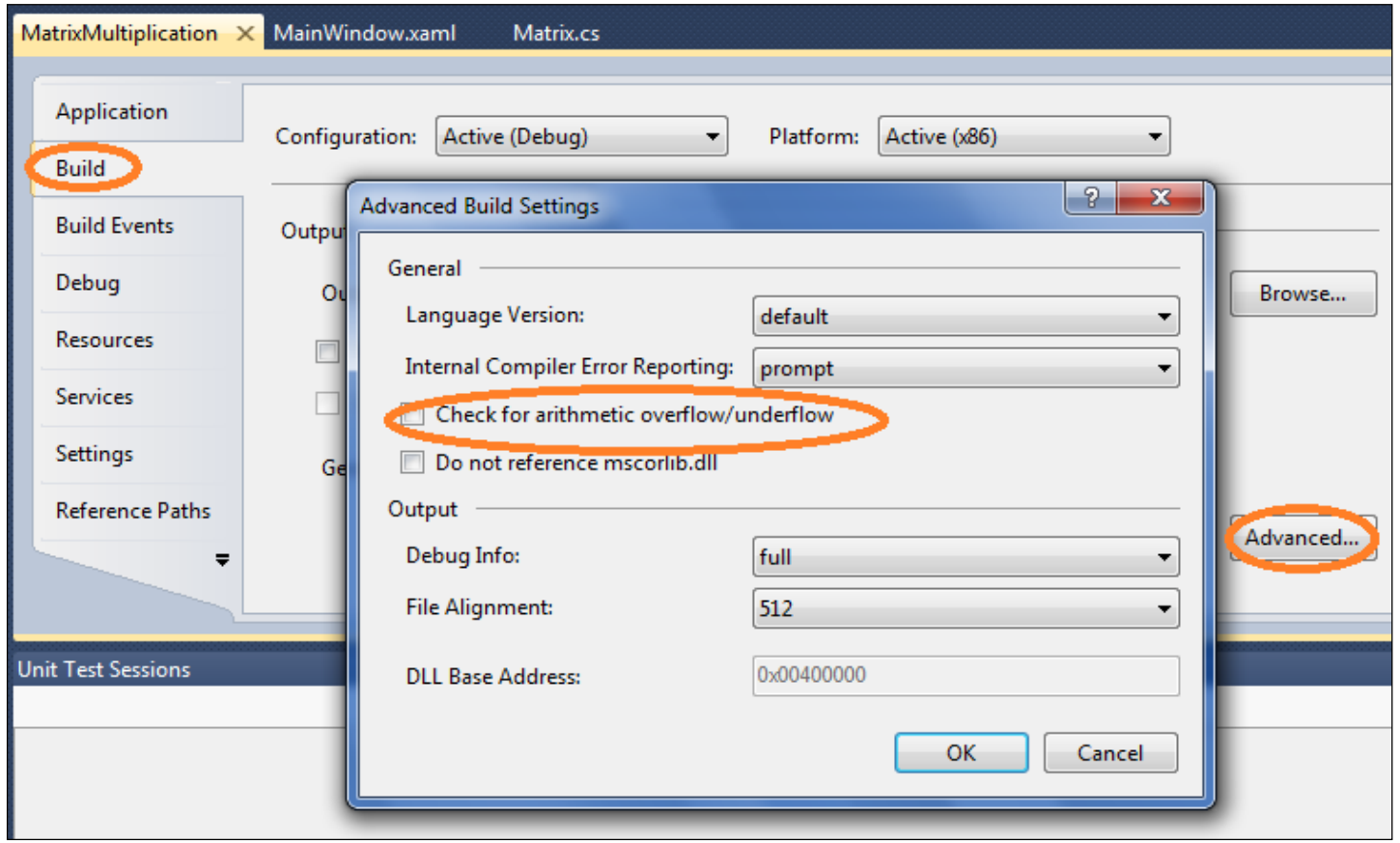
Задача

- Дополнить предыдущий пример обработкой исключений класса Exception

Использование checked и unchecked

- D приложениях Microsoft Visual C# при работе с целыми числами проверка переполнения по умолчанию отключена. В этих приложениях, существует риск того, что числовое переполнение может привести к неправильным результатам. Если существует фрагмент кода, который может привести к числовому переполнению, можно восстановить контроль над ним с помощью ключевого слова checked.
- Включить и отключить проверку переполнения можно также для всего приложения с помощью ключевых слов checked и unchecked соответственно. По умолчанию выражение, содержащее только константные значения, приводит к ошибке компиляции в том случае, если результат его вычисления выходит за допустимые пределы значений конечного типа. Если выражение содержит одно или несколько не константных значений, компилятор не выполняет проверку переполнения.

Использование checked и unchecked



checked

- Ключевое слово checked используется для определения блока кода, заключенного в фигурные скобки, для которого включена проверка численного переполнения. Если в этом блоке происходит переполнение, операторы его вызвавшие приведут к возникновению исключения OverflowException.

Использование checked и unchecked

- checked
- {
- int x = ...;
- int y = ...;
- int z = ...;
- ...
- try
- {
- z = x * y; // May cause numeric overflow
- }
- catch (OverflowException ex)
- {
- ... // Handle the overflow exception
- }
- ...
- }

- public int Multiply(short operandX, short operandY)
- {
- return checked((short)(operandX * operandY));
- }

- unchecked
- {
- int1 = 2147483647 + 10;
- }
- ...
- int1 = unchecked(ConstantMax + 10);
- **Если включена проверка для всего приложения, unchecked необходимо ставить для каждого выражения, чтобы подавить проверку**

Возникновение исключений

- Использование блоков try/catch позволяет приложению перехватывать и обрабатывать исключения. Эти исключения могут быть сгенерированы CLR, если приложение пытается выполнить недопустимые операции.
- В методы можно добавить код, который обнаруживает неправильные условия, а затем генерирует соответствующее исключение, чтобы указать характер ошибки вызывающему метод коду. Тогда вызывающий код сможет перехватить и обработать исключение.

Наиболее распространенные типы исключений

Исключение	Описание
ArgumentException	Исключение создается, если вызывающий код указывает аргумент для метода, не соответствующий требованиям метода. Этот тип исключения можно использовать, чтобы указать обобщенные ошибки с аргументами (можно использовать типы ArgumentOutOfRangeException и ArgumentNullException, чтобы указать более конкретные ошибки).
FormatException	Исключение создается, если вызывающий код специфицирует аргумент, который содержит данные, не имеющие требуемого формата.
NotImplementedException	Исключение создается, чтобы указать, что код еще не имплементирован в методе. Это исключение, в основном, полезно во время разработки кода, когда есть определенный метод, но не написан код тела метода.
NotSupportedException	Исключение создается, если вызывающий код пытается выполнить неподдерживаемые операции с помощью вызываемого метода, например, такие как указание аргументов, определяющих, что абонент хочет писать в файл только для чтения.
ArithmeticException	Основной класс исключений, происходящих при выполнении арифметических операций, таких как DivideByZeroException и OverflowException.
ArrayTypeMismatchException	Исключение создается, когда массив не может хранить данный элемент, поскольку фактический тип элемента несовместим с фактическим типом массива.
DivideByZeroException	Исключение создается при попытке разделить целое число на ноль.
IndexOutOfRangeException	Исключение создается при попытке индексирования массива, когда индекс меньше нуля или выходит за границы массива.
InvalidCastException	Исключение создается, когда во время выполнения происходит сбой явного преобразования из основного типа в интерфейс или производный тип.
NullReferenceException	Исключение создается при попытке ссылки на объект, значение которого равно null.
OutOfMemoryException	Исключение создается при неудачной попытке выделения памяти с помощью оператора new. Это означает, что память, доступная для среды выполнения, уже исчерпана.
OverflowException	Исключение создается при переполнении арифметической операции в контексте checked.
StackOverflowException	Исключение создается, когда стек выполнения переполнен за счет слишком большого количества вызовов отложенных методов; обычно является признаком очень глубокой или бесконечной рекурсии.
TypeInitializationException	Исключение создается, когда статический конструктор создает исключение, и не существует ни одного совместимого предложения catch для его захвата.

Задача

- Дополнить предыдущий пример разной обработкой исключений с использованием других классов, кроме Exception

Создание исключений

- Для создания объекта исключения используется ключевое слово new. При этом следует указать тип исключения и предоставить информацию, которая указывает на причину исключения.
- Как правило, эта информация предоставляется в виде строки, содержащей сообщение об ошибке; в строку можно также включить информацию об еще одном исключении, если объект исключения является результатом этого исключения.
- Текст сообщения об ошибке доступен в свойстве исключения Message в блоке catch, обрабатывающем исключение. Если в исключение включить еще один объект исключения, подробную информацию о нем можно получить в блоке catch, обрабатывающем исключение, в свойстве InnerException.

Пример

```
FormatException ex = new FormatException("Argument has the wrong format");
...
// Example 2
try
{
   ...// Statements that might cause an exception if data is in the wrong format
}
catch (Exception e)
{
   // Create a FormatException containing an error message
   // and a reference to the original exception.
   FormatException ex = new FormatException("Argument has the wrong format", e);
   ...
}
ArgumentException argEx = new ArgumentOutOfRangeException ("param1", "Parameter param1 too large.");
```

Генерация исключений

- После создания объекта исключения, его можно генерировать (throw), чтобы указать, что произошло исключение.
- **throw [exception object];**
- Когда генерируется исключение, исполнение текущего блока кода прекращается, и CLR передает управление первому доступному обработчику исключений, который перехватывает исключение так, как это было описано ранее. Генерация исключения является дорогостоящей операцией с точки зрения циклов процессора (CPU), поэтому ее следует использовать с осторожностью.

- FormatExeption ex = new FormatExeption("Argument has the wrong format");
- throw ex;

Генерация исключений

- Если блок catch для исключения не может устранить ошибку, он может перебросить исключение и распространить его на вызывающий код.
- try
- {
- ... // Statements that might cause an exception
- }
- catch(Exception e)
- {
- // Attempt to handle the exception
- ...
- // If this catch handler cannot resolve the exception,
- // throw it to the calling code
- throw;
- }

Когда генерировать исключения

- *Метод не способен выполнить свои определенные функции*. Например, если значение параметра метода является недопустимым.
- *На основе состояния объекта выполнен неправильный вызов объекта*. В качестве примера можно привести попытку записи в файл, доступный только для чтения. В случаях, когда состояние объекта не допускает выполнения операции, генерируется экземпляр InvalidOperationException или объекта на основе наследования этого класса.
- *Аргумент метода вызывает исключение*. В этом случае, должно быть перехвачено исходное исключение и создан экземпляр ArgumentException. Исходное исключение должно быть передано конструктору ArgumentException в качестве параметра InnerException.

Задача

- Дополнить предыдущий пример явной генераций исключения

Рекомендации по обработке и генерации исключений

- Следует генерировать исключение, соответствующее обнаруженному ошибочному условию.
- Логика приложения не должна полагаться на логику блоков try и catch для работы в рамках неисключительных условий. Следует разрабатывать методы так, чтобы при нормальных обстоятельствах они не выбрасывали исключений. Создавать и генерировать исключения нужно для условий, которые находятся вне ожидаемого логического потока приложения.
- При определении нескольких блоков catch, следует упорядочивать их спецификации от наиболее конкретных к наименее конкретным. Если перехватывается тип исключения Exception, то он должен быть последним обработчиком во множестве блоков catch.
- Следует перехватывать исключения и детализировать сообщения для диагностических целей, а затем отображать удобные для пользователя сообщения. Любой текст, который отображается для пользователя должен быть локализуемым, текст должен извлекаться из файлов ресурсов.
- Не желательно показывать пользователю слишком подробную информацию об исключении, поскольку она может быть использована злонамеренными пользователями для вывода приложения из строя или получения доступа к защищенной информации. Например, распространенной ошибкой, выполняемой в слоях доступа к данным, является представление подробной информации об ошибке в результате неправильного запроса к базе данных. Такая информация может позволить злоумышленнику понять логику приложения и использовать эти знания для атаки на систему.
- Эффективная обработка исключения должна позволить приложению восстановиться после возникновения исключения, а пользователю – продолжить пользоваться приложением. В случае возникновения исключения пользователь не должен потерять данные, а приложение не должно потерпеть крах.

Генерация исключений

- Если исключение сгенерировано в финализаторе, то обработчик исключений может не знать о том, как его обрабатывать и прервет работу приложения.
- Если исключение сгенерировано в статическом конструкторе, то этот конструктор не выполнится до конца, и класс будет недоинициализирован.

Пользовательские исключения

- public class InvalidDepartmentException : System.Exception
- {
- public InvalidDepartmentException() : base() { }
- public InvalidDepartmentException(string message) : base(message) { }
- public InvalidDepartmentException (string message, System.Exception inner)
- : base(message, inner) { }
- }

Задача

- Предыдущую задачу дополнить собственным классом обработки исключений в операциях над отрезками

Домашнее задание

- Дополнить предыдущее задание следующим функционалом:
- Перегрузить операторы для сравнения фигур и математических операций (с разными типами). Использовать операции преобразования
- Создать свой класс исключений, который будет генерировать исключения в случае, если невозможно выполнить операции с фигурами. Предусмотреть возможные исключения для универсального класса.