# ПРАКТИЧЕСКАЯ РАБОТА № 12

**Обработка исключительных ситуаций**

**Цель работы**: научиться выявлять исключительные ситуации и обрабатывать их.

|  |  |  |
| --- | --- | --- |
| №  п/п | Ф.И.О. | НОМЕР ВАРИАНТА/ЗАДАНИЯ |
|  | Андриуца Дарья Николаевна | 1 |
|  | Варфоломеев Иван Константинович | 2 |
|  | Верховин Игорь Вадимович | 3 |
|  | Власов Тимур Михайлович | 4 |
|  | Гагауз Григорий Иванович | 5 |
|  | Голубцов Олег Русланович | 6 |
|  | Гувирь Максим Григорьевич | 7 |
|  | Дорожкина Алина Игоревна | 8 |
|  | Жупанов Юлиан Игоревич | 9 |
|  | Капелько Иван Юрьевич | 10 |
|  | Кирста Андрей Александрович | 11 |
|  | Ковалев Иван Алексеевич | 12 |
|  | Лютов Глеб Васильевич | 13 |
|  | Михалаки Василий Федорович | 14 |
|  | Москалюк Иван Сергеевич | 15 |
|  | Огурцов Алексей Михайлович | 1 |
|  | Омельченко Евгений Борисович | 2 |
|  | Полякова Анна Игоревна | 3 |
|  | Редкозубов Владислав Романович | 4 |
|  | Ротарь Федор Федорович | 5 |
|  | Рошко (был Брага) Марин Михайлович | 6 |
|  | Салкуцан Анна Александровна | 7 |
|  | Скутельник Владимир Александрович | 8 |
|  | Струсевич Лев Дмитриевич | 9 |
|  | Телицкий Александр | 10 |
|  | Чернюк Егор Дмитриевич | 11 |
|  | Щитченко Николь Александровна | 12 |

# Теоретические сведения

Какой бы надежный код ни был написан, сколь бы тщательной ни была отладка, в версии, переданной в эксплуатацию и на сопровождение, при запусках будут встречаться нарушения спецификаций. В системе остается последняя ошибка, находятся пользователи, не знающие спецификаций, и если спецификацию можно нарушить, то это событие когда-нибудь да произойдет. В таких **исключительных ситуациях** продолжение выполнения программы либо становится невозможным, либо в возникшей ситуации применение алгоритма приведет к ошибочным результатам.

Рассмотрим обработки исключений, т.е. конструкцию *try-catch-finally*:

*try {...}*

*catch (T1 e1) {...}*

*...*

*catch(Tk ek) {...}*

*finally {...}*

Любой набор команд можно объявить охраняемым блоком. Для этого необходимо использовать ключевое слово *try*. Вслед за *try*-блоком могут следовать *catch*-блоки, называемые **блоками-обработчиками исключительных ситуаций**, их может быть несколько, они могут и отсутствовать. Завершает эту последовательность *finally***-**блок - блок финализации, который также может отсутствовать. Вся эта конструкция может быть вложенной - в состав *try*-блока может входить конструкция *try*- *catch*-*finally*.

В теле try-блока может возникнуть исключительная ситуация, приводящая к **выбрасыванию исключений**. Выбрасывание исключения происходит при выполнении оператора *throw*. Этот оператор может быть частью программного текста *try*-блока и выполняться, когда в результате

проведенного анализа становится понятным, что дальнейшая нормальная работа невозможна.

Синтаксически оператор *throw* имеет вид: *throw***[**выражение**]**

Выражение *throw* задает объект класса, являющегося наследником класса *Exception*. Обычно это выражение *new*, создающее новый объект. Если оно отсутствует, то повторно выбрасывается текущее исключение. Если исключение выбрасывается операционной системой, то она сама классифицирует исключение, создает объект соответствующего класса и автоматически заполняет его поля.

В рассматриваемой нами модели исключения являются объектами, класс которых представляет собой наследника класса *Exception*. Этот класс и многочисленные его наследники является частью библиотеки *FCL*, хотя и разбросаны по разным пространствам имен. Каждый класс задает определенный тип исключения в соответствии с классификацией, принятой в *Framework*.*Net*.

Рассмотрим примеры классов исключений из пространства имен

System:

*Argument Exception, ArgumentOutOfRangeException, ArithmeticException, BadImageFormatException, DivideByZeroException, OverflowException.*

В пространстве имен *System*.*IO* собраны классы исключений, связанных с проблемами ввода-вывода: *DirectoryNotFoundException, FileNotFoundException* и многие другие.

Имена всех классов исключений заканчиваются словом *Exception*. Разрешается создавать собственные классы исключений, наследуя их от класса *Exception*.

При выполнении оператора *throw* создается объект *te*, класс *TE* которого характеризует текущее исключение, а поля содержат информацию о возникшей исключительной ситуации. Выполнение оператора *throw* приводит к тому, что нормальный процесс вычислений на этом прекращается. Если это происходит в охраняемом *try*-блоке, то начинается

этап **"***захвата***"** исключения одним из обработчиков исключений.

Блок *catch* - обработчик исключения имеет следующий синтаксис:

*catch (T e) {...}*

Класс *T*, указанный в заголовке *catch*-блока, должен принадлежать классам исключений. Блок *catch* с формальным аргументом *e* класса *T* потенциально способен захватить текущее исключение *te* класса *TE*, если и только если объект *te* совместим по присваиванию c объектом *e*. Другими словами, потенциальная способность захвата означает допустимость присваивания *e = te*, что возможно, когда класс *TE* является потомком класса

*T*. Обработчик, класс *T* которого является классом *Exception*, является универсальным обработчиком, потенциально он способен захватить любое исключение, поскольку все они являются его потомками.

Потенциальных захватчиков может быть много, исключение захватывает лишь один - тот из них, кто стоит первым в списке проверки. Рассмотрим порядок проверки. Вначале проверяются обработчики в порядке следования их за *try*-блоком, и первый потенциальный захватчик становится активным, захватывая исключение и выполняя его обработку. Отсюда следует, что порядок следования в списке *catch*-блоков крайне важен.

Первыми идут наиболее специализированные обработчики, далее по мере возрастания универсальности. Так, вначале должен идти обработчик исключения *DivideByZeroException*, а уже за ним – *ArithmeticException*. Универсальный обработчик, если он есть, должен стоять последним.

Необходимо помнить, что *try*-блок может быть вложен в другой *try*- блок. Когда же будет исчерпаны списки вложенных блоков, а потенциальный захватчик не будет найден, то произойдет подъем по стеку вызовов. На рисунке 11.1 показана цепочка вызовов, начинающаяся с процедуры *Main*.

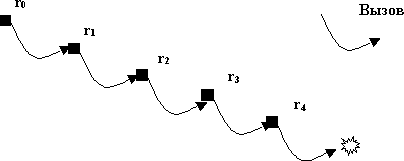


Рисунок 11.1. Цепочка вызовов, хранящаяся в стеке вызовов

Исключение возникло в последнем вызванном методе цепочки - на рисунке метод *r5*. Если у этого метода не нашлось обработчиков события, способных обработать исключение, то это пытается сделать метод *r4*, вызвавший *r5*. Если вызов *r5* находится в охраняемом блоке метода *r4*, то начнет проверяться список обработчиков в охраняемом блоке метода *r4*. Этот процесс подъема по списку вызовов будет продолжаться, пока не будет найден обработчик, способный захватить исключение, или не будет достигнута начальная точка - процедура *Main*. Если и в ней нет потенциального захватчика исключения, то сработает стандартный обработчик, прерывающий выполнение программы с выдачей соответствующего сообщения.

Обработчику исключения - *catch*-блоку, захватившему исключение, - передается текущее исключение. Анализируя свойства этого объекта, обра- ботчик может понять причину, приведшую к возникновению **исключитель- ной ситуации**, попытаться ее исправить и в случае успеха продолжить вы- числения. В принятой C# **схеме без возобновления** обработчик исключения не возвращает управление *try*-блоку, а сам пытается решить проблемы. После завершения *catch*-блока выполняются операторы текущего метода, следу- ющие за конструкцией *try-catch-finally*.

Зачастую обработчик исключения не может исправить ситуацию или может выполнить это лишь частично, предоставив решение оставшейся части проблем вызвавшему методу - предшественнику в цепочке вызовов.

Механизм, реализующий такую возможность - это тот же механизм исключений. Как правило, в конце своей работы обработчик исключения

выбрасывает исключение, выполняя оператор *throw*. При этом у него есть две возможности: повторно выбросить текущее исключение или выбросить новое исключение, содержащее дополнительную информацию.

Таким образом, обработку возникшей исключительной ситуации могут выполнять несколько обработчиков, принадлежащие разным уровням цепочки вызовов.

Блок *finally* - важный участник схемы обработки исключений. Мы рассматриваем схему без возобновления. Это означает, что управление вычислением неожиданно покидает *try*-блок. Как правило, при этом необходимо выполнить некоторые действия, например, выполнить определенную чистку. Прежде всего, удаляются все локальные объекты, созданные в процессе работы блока. В C#, благодаря автоматической сборке мусора, освобождением памяти можно не заниматься, достаточно освободить стек. Но в блоке *try* могли быть заняты другие ресурсы - открыты файлы, захвачены некоторые устройства. Освобождение ресурсов, занятых *try*- блоком, выполняет *finally*-блок. Если он присутствует, он выполняется всегда, сразу же после завершения работы *try*-блока, как бы последний ни завершился. Блок *try* может завершиться вполне нормально без всяких происшествий и управление достигнет конца блока, выполнение может быть прервано оператором *throw*, управление может быть передано другому блоку из-за выполнения таких операторов как *goto, return* - во всех этих случаях, прежде чем управление будет передано по предписанному назначению (в том числе, прежде чем произойдет захват исключения), предварительно будет выполнен *finally*-блок, который освобождает ресурсы, занятые *try*-блоком, а параллельно будет происходить освобождение стека от локальных переменных.

Схема обработки исключительных ситуаций, предложенная в языке C#,

обладает одним существенным недостатком - ее можно применить некорректно. Она позволяет, в случае возникновения исключительной ситуации, уведомить о ее возникновении и спокойно продолжить работу, что,

в конечном счете, приведет к неверным результатам. Из двух зол - прервать вычисление с уведомлением о невозможности продолжения работы или закончить вычисления с ошибочным результатом вычисления - следует выбирать первое. Некорректно примененная схема C# приведет к ошибочным результатам.

Приведем пример. Пусть необходимо оформить документы для турпоездки – билеты туда и обратно, забронировать место в гостинице. В ходе выполнения этих действий возникает исключительная ситуация - нет свободных билетов обратно, при этом обработчик исключения посылает уведомление с принесением извинений, но оформление турпоездки продолжается.

Бертран Мейер предложил следующую схему обработки исключительных ситуаций. В основе ее лежит подход к проектированию программной системы на принципах Проектирования по Контракту. Модули программной системы, вызывающие друг друга, заключают между собой контракты. Вызывающий модуль обязан обеспечить истинность предусловия, необходимого для корректной работы вызванного модуля.

Вызванный модуль обязан гарантировать истинность постусловия по завершении своей работы. Если в вызванном модуле возникает исключительная ситуация, то это означает, что он не может выполнить свою часть контракта. Какие действия должен выполнить обработчик исключительной ситуации? У него только две возможности *- Retry* и *Rescue*.

Первая (*Retry*) - попытаться внести некоторые коррективы и вернуть управление охраняемому модулю, который может предпринять очередную попытку выполнить свой контракт. Модуль может, например, в следующей попытке запустить другой алгоритм, использовать другой файл, другие данные. Если все закончится успешно, и работа модуля будет соответствовать его постусловию, то появление исключительной ситуации можно рассматривать как временные трудности, успешно преодоленные.

В том случае, если ситуация возникает вновь и вновь, то обработчик

исключения применяет вторую стратегию (*Rescue*), выбрасывая исключение и передавая управление вызывающему модулю, который и должен теперь попытаться исправить ситуацию. Важная тонкость в схеме, предложенной Бертраном, состоит в том, что исключение, выбрасываемое обработчиком, следует как отход на заранее подготовленные позиции. Обработчик исключения должен позаботиться о восстановлении состояния, предшествующего вызову модуля, который привел к исключительной ситуации, и это гарантирует нахождение всей системы в корректном состоянии.

Одной из наиболее удачных схем обработки исключений является Схема Бертрана. Рассмотрим возможную реализацию этой схемы на C#:

*public void Pattern()*

*{*

*do {*

*try {*

*bool Danger = false;*

*Success = true;*

*MakeJob();*

*Danger = CheckDanger(); if (Danger)*

*throw (new MyException()); MakeLastJob();*

*}*

*catch (MyException me)*

*{*

*if(count > maxcount)*

*throw(new MyException("все попытки были безуспешны")); Success = false; count++;*

*//корректировка ситуации Console.WriteLine("Попытка что-то изменить!"); level +=1;*

*}*

*}while (!Success);*

*}*

Краткие комментарии:

* Конструкция *try-catch* блоков помещается в цикл do-*while*(!

*Success*), завершаемый в случае успешной работы охраняемого блока, за чем следит булева переменная *Success*.

* В данном образце предполагается, что в теле охраняемого блока анализируется возможность возникновения исключительной ситуации и, в

случае обнаружения опасности, выбрасывается собственное исключение, класс которого задан программно. В соответствии с этим тело try-блока со- держит вызов метода *MakeJob*, выполняющего некоторую часть работы, по- сле чего вызывается метод *CheckDanger*, выясняющий, не возникла ли опас- ность нарушения спецификации и может ли работа быть продолжена. Если все нормально, то выполняется метод *MakeLastJob*, выполняющий заключи- тельную часть работы. Управление вычислением достигает конца *try*-блока, он успешно завершается и, поскольку остается истинной переменная *Success*, значение *true* которой установлено в начале *try*-блока, то цикл *while*, окайм- ляющий охраняемый блок и его обработчиков исключений, также успешно завершается.

* Если в методе *CheckDanger* выясняется, что нормальное продол- жение вычислений невозможно, то выбрасывается исключение класса *MyException*. Оно перехватывается обработчиком исключения, стоящим за *try*-блоком, поскольку класс *MyException* указан как класс формального аргумента.
* Для простоты приведем только один *catch*-блок. В общем случае их может быть несколько, но все они строятся по единому образцу. Предпо- лагается, что обработчик исключения может сделать несколько попыток ис- править ситуацию, после чего повторно выполняется охраняемый блок. Если же число попыток, за которым следит переменная *count*, превосходит мак- симально допустимое, то обработчик выбрасывает новое исключение, зада- вая дополнительную информацию и передавая тем самым обработку ошибки на следующий уровень - вызываемой программе.
* Когда число попыток еще не исчерпано, обработчик исключения переменной *Success* дает значение *false*, гарантирующее повтор выполнения *try*-блока, увеличивает счетчик числа попыток и пытается исправить ситуа- цию.
* Данная эта схема реализует два корректных исхода обработки ис-

ключительной ситуации - *Retry* и *Rescue* - повтор с надеждой выполнить обя-

зательства и передачу управления вызывающей программе, чтобы она предприняла попытки исправления ситуации, когда вызванная программа не смогла с этим справиться.

Доведем этот образец до реально работающего кода, где угроза исключения зависит от значения генерируемого случайного числа, а обработчик исключения может изменять границы интервала, повышая вероятность успеха.

Определим первым делом собственный класс исключений:

*public class MyException :Exception*

*{*

*public MyException() {}*

*public MyException (string message) : base(message) {}*

*public MyException (string message, Exception e) : base(message, e) {}*

*}*

Минимум того, что нужно сделать, определяя свои исключения, - это

задать три конструктора класса, вызывающие соответствующие конструкторы базового класса *Exception*.

В классе *Excepts*, методом которого является наш образец *Pattern*,

определим следующие поля класса:

*class Excepts*

*{*

*Random rnd = new Random(); int level = -10;*

*bool Success; //true - нормальное завершение int count =1; // число попыток выполнения const int maxcount =3;*

*void MakeJob(){*

*Console.WriteLine("Подготовительные работы завершены");*

*}*

*bool CheckDanger()*

*{*

*//проверка качества и возможности продолжения работ*

*int low = rnd.Next(level,10); if ( low > 6) return(false); return(true);*

*}*

*void MakeLastJob(){*

*Console.WriteLine("Все работы завершены успешно"); }*

*}*

Зададим метод, вызывающий метод *Pattern*:

*public void TestPattern()*

*{*

*Excepts ex1 = new Excepts(); try {*

*ex1.Pattern();*

*}*

*catch (Exception e)*

*{*

*Console.WriteLine("исключительная ситуация при вызове Pattern"); Console.WriteLine(e.ToString());*

*}*

*}*

Обратите внимание, что вызов метода *Pattern* находится внутри охраняемого блока. Поэтому, когда *Pattern* не справится с обработкой исключительной ситуации, ее обработку возьмет на себя универсальный обработчик, стоящий за try-блоком.

Рассмотрим устройство базового класса *Exception*.

Основными свойствами класса являются:

* *Message* - строка, задающая причину возникновения исключения. Зна- чение этого свойства устанавливается при вызове конструктора класса, когда создается объект, задающий исключение;
* *HelpLink* - ссылка (*URL*) на файл, содержащий подробную справку о возможной причине возникновения исключительной ситуации и способах ее устранения;
* *InnerException* - ссылка на внутреннее исключение. Когда обработчик выбрасывает новое исключение для передачи обработки на следующий уровень, то текущее исключение становится внутренним для вновь создава- емого исключения;
* *Source* - имя приложения, ставшего причиной исключения;
* *StackTrace* - цепочка вызовов - методы, хранящиеся в стеке вызовов в момент возникновения исключения;
* *TargetSite* - метод, выбросивший исключение.

Рассмотрим метод класса *GetBaseException*. При подъеме по цепочке вызовов он позволяет получить исходное исключение - первопричину возникновения последовательности выбрасываемых исключений.

Класс имеет четыре конструктора. Один из них - конструктор без

аргументов, второй - принимает строку, становящуюся свойством *Message*, третий - имеет еще один аргумент: исключение, передаваемое свойству *InnerException*.

Внесем в предыдущий пример ряд изменений, а именно добавим еще один аргумент при вызове конструктора исключения в *catch*-блоке метода *Pattern*: *throw*(*new MyException*("Все попытки *Pattern* безуспешны", me));

В этом случае у создаваемого исключения заполняется свойство *InnerExceptions*. Для слежения за свойствами исключений добавим метод печати всех свойств, вызываемый во всех обработчиках исключений:

*static public void PrintProperties(Exception e)*

*{*

*Console.WriteLine("Свойства исключения:"); Console.WriteLine("TargetSite = {0}", e.TargetSite); Console.WriteLine("Source = {0}", e.Source); Console.WriteLine("Message = {0}",e.Message);*

*if (e.InnerException == null) Console.WriteLine("InnerException = null"); else Console.WriteLine("InnerException = {0}", e.InnerException. Message); Console.WriteLine("StackTrace = {0}", e.StackTrace); Console.WriteLine("GetBaseException = {0}", e.GetBaseException());*

*}*

Следует помнить о двух важных правилах:

* + обработка исключений должна быть направлена не столько на

уведомление о возникновении ошибки, сколько на корректировку возникшей ситуации;

* + если исправить ситуацию не удается, то программа должна быть прервана так, чтобы не были получены некорректные результаты, не удовле- творяющие спецификациям программы.

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОГО ВЫПОЛНЕНИЯ

Разработать программу, выбрасывающую исключение и использующую не менее 3-х обработчиков:

1. Студент знает ответы на 10 экзаменационных вопросов из 30. Если заданный вопрос ему не известен, то он: а) пытается вспомнить; б) пробует воспользоваться шпаргалкой; в) несет околесицу.
2. Студент ждет свою маршрутку на остановке. Всего есть 10

маршрутов, из которых его устраивают только 2.

Если попадается нужная маршрутка, он едет на занятие, если не подходящая маршрутка, то он ждет следующую. Если он пропустил больше 3 маршруток, то он звонит другу, что опаздывает на занятие, если больше 5 – то идет домой.

1. Есть 3 студента и 3 студентки, между которыми имеются определенные взаимоотношения: симпатия, нейтральное, отвращение. Один из студентов приглашает наугад выбранную студентку в кино. Если в отношениях симпатия, то она соглашается, если отношения нейтральны, то вежливо отказывает, если отвращение – то не замечает приглашения.

Если девушка не соглашается, то она ждет приглашения от другого студента. Допускается 3 попытки.

1. Маршрутное такси едет по маршруту. На остановках садятся и выходят пассажиры. Иногда такси останавливает ГАИ. Если в маршрутке пассажиров больше, чем n, то выписывается штраф, если больше чем m, то отбирают права. Если причина остановки – техническая, то такси дальше не идет. Всего на маршруте 5 остановок.
2. Лектор читает лекцию перед группой из 25 студентов. Часть студентов разговаривает. Если их не более 4%, то преподаватель не обращает внимания, если больше 10%, то делает им замечание и продолжает лекцию, если их не более 20%, то устраивает им разнос, если и это не помогает, то приводит заместителя декана.
3. Автомобиль движется по шоссе, преодолевая маршрут из 200 км. В пути возможны следующие ситуации: кончился бензин, водитель устал, лопнула шина, встретил ГАИ.
4. Компьютер может выйти из строя по следующим причинам: отказ

«железа», неправильные действия оператора, сбой в операционной системе, перегрев. Написать программу, имитирующую жизнь компьютера в течение некоторого времени.

1. Студент сдает сессию из 5 экзаменов. Если он не сдал >=3

экзаменов, то его или отчисляют или он переводится на курс ниже, если не сдал 1-2 экзамена, то ему назначают переэкзаменовку. Учесть, что студент может сдавать экзамен по одному предмету не более 3-х раз.

1. Водитель едет по трассе. Если он совершил ДТП с незначительными повреждениями автомобиля, то его путешествие задерживается на 2 час – для составления протокола, если ДТП с серьезными повреждениями, то вызывают эвакуатор, если просто нарушение правил дорожного движения, то штрафуют. Учесть, что после 3-го штрафа у водителя забирают права.
2. Подчиненный получает задание руководителя и пытается выполнить его. Если ему это удается, то он получает новое задание. Если задание не выполнено, то руководитель штрафует сотрудника и дает новое задание. Учесть, что после 3-х подряд не выполненных заданий, сотрудника увольняют, а после 10 подряд выполненных – повышают по службе. Предусмотреть выполнение 50 заданий
3. Сотрудник офиса работает с поступившими документами, оставляя у себя, их копии. Если на столе хранится более n документов, то он перекладывает их в ящики стола, и складывает новые документы, если в ящиках стола хранится более m документов, то сдает их в архив. Если сотрудника увольняют, то он удаляет все документы из своего стола.
4. Программа обрабатывает полученный набор данных. Если в процессе обработки выявляется ошибка (например, текущее число больше максимально возможного), то программа заменяет его на среднее арифметическое следующих 3- чисел, если полученное число все равно больше максимально возможного, то программа меняет его на среднее следующих 10 чисел, если число и после этого слишком велико, то его меняют на среднее по всей выборке, если и это не решает проблему, то программа выдает сообщение и прекращает свою работу.
5. Программа ищет среди студентов группы того, чьи оценки, по итогам сессии, в среднем >4.5 (при отсутствии двоек и задолженностей), если такого студента нет, то производится поиск студента, чей средний балл

>4 (при отсутствии двоек и задолженностей), если и таких студентов нет, то ищется студент, у которого оценки по математике =5 и по информатике =5, если таких нет то выдается сообщение и программа прекращает свою работу.

1. Сотрудник фирмы пытается снять квартиру. Если стоимость съема предложенной квартиры >*Rmax*, то объявляется повторный поиск, причем разрешенный порог стоимость увеличивается на 10%, если и в этом случае предложенные квартиры слишком дороги, то порог стоимости увеличивается еще на 10%, если и это не помогает, то сотрудник увольняется.
2. Фирма пытается заключить контракт на поставку некоторого списка товаров.

При этом формируется список из 3-х товаров, для каждого товара указывается количество и цена. Если предложенные фирмами - поставщиками цены и количество товаров не фирму-покупателя не устраивают, то цены товаров на 10% уменьшаются и конкурс повторяется, если цена товара покупателя устроила, а количество – меньше, чем требуется, то заключается контракт на указанное поставщиком количество товара, в противном случае выдается сообщение, что закупить товар не удалось.