Deakin University

Object Oriented Development

OnTrack Submission

Exceptions and Error Handling

Submitted By:
Peter Stacey
pstacey
2020/04/14 14:58

Tutor:
Dipto Pratyaksa

Outcome	Weight
Evaluate Code	♦♦♦◊◊
Principles	♦♦♦♦◊
Build Programs	♦♦♦◊◊
Design	♦♦◊◊◊
Justify	♦♦♦♦♦

This task, which involves both implementing code to specifically thrown exceptions, and researching those exceptions to identify their uses, who should thrown them, whether they can be caught, what information to provide and how to avoid them, is a good example through the evidence in the report, of justifying findings with references and evidence. Additionally, the program, which implements the exceptions also demonstrates the ability to follow conventions to a specific outcome, in this case, a set of exceptions that are handled wherever possible.

April 14, 2020

SIT232 – Object Oriented Development Task 4.1P- Report on Exceptions in C#

Student Name: Peter Stacey

Student ID: 219011171

Introduction to Exceptions

Exception handling is an important aspect of developing resilient programs that meet user needs. It helps to identify and respond to exceptional circumstances during the runtime of a program, and allows these exceptional circumstances to be handled gracefully, ideally with no interruption visible to the end-user.

In this task we look at a sub-set of the large number of exceptions that derive from the Exception class (refer Figure 1).

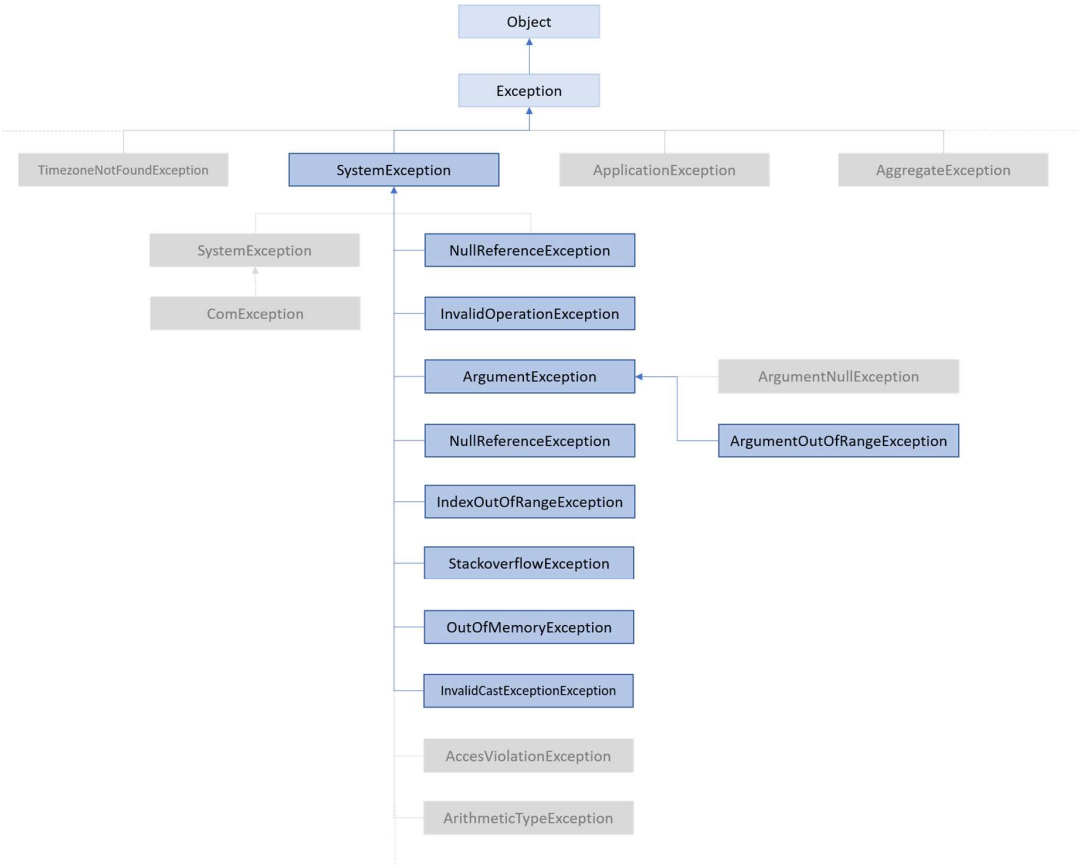

Figure 1: Hierarchy of Exceptions investigated

NullReferenceException

Possible situation that leads to the exception	Occurs when trying to access a member on a type whose value is null
Who is in charge of throwing the exception?	The called code, where it relies on arguments not being null. If it is internally used code, then the exception should be eliminated, but where arguments or types are passed to a piece of code, that code should call the exception if it relies on the data not being null. Null is unusual in that it is common for types to be null at different times, in different programs. Some programmers intentionally use and pass around null references for different uses. For example, in C# database tables can have nullable fields to indicate there is no value stored. If code then accesses that database and wants to operate on the data, checking for null types is important, to handle the exception locally in the code, or pass the exception to the calling point.
What details would you provide to the callers when throwing this exception?	• The type of the exception • Which type or argument is null • Inform them they may have forgotten to initialize the type
Can the exception be generally caught?	Yes, this can be generally caught when it is thrown
Should you catch this exception type or pass to the user?	If the exception results inside my own code, I would eliminate the risk of it all together by properly initializing types that I use. If the type is passed by a caller into a method, then I would pass it back to the caller.
Is the exception a case when you want to avoid the exception to occur? If so, what would be your actions as a programmer to avoid it?	Yes, in my own code, I would always want to avoid this error by ensuring that all variables and objects are properly initialized before using them. However, since I can't control the values passed by users of my code, I would ensure that I can catch it where it needs to be and pass the exception to the caller.

References

https://docs.microsoft.com/en-us/dotnet/api/system.nullreferenceexception?view=netframework-4.8

https://stackoverflow.com/questions/4660142/what-is-a-nullreferenceexception-and-how-do-i-fix-it

IndexOutOfRangeException

Possible situation that leads to the exception	When attempting to access an element of an array or collection with an index that is outside its bounds. That is, either an index that is below the lowest index, or an index greater than the maximum index.
Who is in charge of throwing the exception?	The code that tries to access an index that is not in the available range.
What details would you provide to the callers when throwing this exception?	• The type of the exception • If there is an acceptable range, the range of values that can be used for the index or method • If there isn't a known acceptable range (eg. In C++, ranging over an array requires the length of the array to be passed in as there is no equivalent to C# Length property in C++), then I would let them know that the index is outside the range
Can the exception be generally caught?	Yes, this can be generally caught when it is thrown
Should you catch this exception type or pass to the user?	The called code should catch and handle this wherever possible, so that there is no further action required by the calling point.
Is the exception a case when you want to avoid the exception to occur? If so, what would be your actions as a programmer to avoid it?	Yes, this can in most cases be avoided by using methods and properties that return the length of an array or collection, and then the values can be used. However, in cases such as the MyTime class we created for Task 2.3C, we would want to explicitly call the exception when an argument outside the allowed range is supplied.

References

https://docs.microsoft.com/en-us/dotnet/api/system.indexoutofrangeexception?view=netframework-4.8

https://stackoverflow.com/questions/20940979/what-is-an-indexoutofrangeexception-argumentoutofrangeexception-and-how-do-i-f

StackOverflowException

Possible situation that leads to the exception	Thrown when the execution stack overflows because it contains too many nested method calls.
Who is in charge of throwing the exception	The Common Language Runtime or System throws this exception.
What details would you provide to the callers when throwing this exception	Standard output is a “Stack overflow” message to the standard output.
Can the exception be generally caught	No. Starting in .NET Framework 2.0, the exception cannot be caught with a try/catch block and the process terminates by default.
Should you catch this exception type or pass to the user	There is no option. The process will terminate by default. According to the C# documentation, even applying HandleProcessCorruptedStateExceptionsAttribute will have no effect when this exception occurs.
Is the exception a case when you want to avoid the exception to occur? If so, what would be your actions as a programmer to avoid it?	Yes. Code should be written to detect and prevent possible stack overflow, since the effect is termination of the process. This is particularly relevant for situations that involve recursion. It is possible in the CLR to specify that the CLR should unload the application domain that caused the exception and let the process continue, however it is better to prevent the possibility from occurring. As per the Stackoverflow link below, which contains a code example, there are a couple of approaches: • Write code that checks the xsl for infinite recursion and notifies the user prior to applying a transform. • Load the XslTransform code into a separate process.

References

https://docs.microsoft.com/en-us/dotnet/api/system.stackoverflowexception?view=netframework-4.8

https://stackoverflow.com/questions/206820/how-do-i-prevent-and-or-handle-a-stackoverflowexception

OutOfMemoryException

Possible situation that leads to the exception	In general, the exception is thrown when there is not enough memory to continue execution of a program. There are two major causes: 1. Attempting to expand a StringBuilder object beyond the maximum capacity defined by the MaxCapacity property 2. When the CLR cannot allocate enough contiguous memory to full support an operation Four examples of the second case include: • When a disk is physically full and/or signficiantly fragmented, so that there are few contiguous blocks of memory left, • Running a program on a 32-bit operating system, which is limited to 2 GB memory, but expecting to be able to use more, • Working in a Big Data environment with a large volume of data that the system cannot manage effectively, or • Memory leaks.
Who is in charge of throwing the exception?	The system in terms of the CLR will throw the exception.
What details would you provide to the callers when throwing this exception?	• The exception type • The process that was running when the exception occurred • Hint to check physical disks and defrag their environment
Can the exception be generally caught?	Yes, this can be generally caught, and a strategy developed to handle it gracefully.
Should you catch this exception type or pass to the user?	Catch and handle if possible. Otherwise if it cannot be handled, pass it to the caller.
Is the exception a case when you want to avoid the exception to occur? If so, what would be your actions as a programmer to avoid it?	Yes, if possible. It is good to avoid (eg. Use generators to limit the memory use of large datasets), but it may not be possible (eg. System physical memory is nearly full).

References

https://docs.microsoft.com/en-us/dotnet/api/system.outofmemoryexception?view=netframework-4.8

https://stackoverflow.com/questions/8563933/c-sharp-out-of-memory-exception

InvalidCastException

Possible situation that leads to the exception	When trying to cast a type to another type that it cannot be cast to. For example, casting a boolean value to a char type, or casting an object to a inbuilt type (eg. int)
Who is in charge of throwing the exception	The CLR will throw this if it cannot cast one type to another and the programmer pass this back to the caller, as there is no real strategy to deal with it effectively.
What details would you provide to the callers when throwing this exception	• The type of the exception • Which cast failed • The type of the input and the type trying to cast to
Can the exception be generally caught	Yes, this can be caught, although handling it gracefully without passing it to the user is difficult.
Should you catch this exception type or pass to the user	Pass this to the user, so they can correct the issue before calling the method that causes the exception, or develop a different strategy in their software.
Is the exception a case when you want to avoid the exception to occur? If so, what would be your actions as a programmer to avoid it?	Yes, since dealing with it after the exception is thrown is more difficult. Ideally, check all variables that need to be cast, to ensure they can be cast to relevant types and/or avoiding the need to cast wherever possible.

References

https://docs.microsoft.com/en-us/dotnet/api/system.invalidcastexception?view=netframework-4.8

https://stackoverflow.com/questions/14327071/how-do-i-solve-an-invalidcastexception/14327121

https://stackoverflow.com/questions/39891504/casting-object-to-int-throws-invalidcastexception-in-c-sharp

DivideByZeroException

Possible situation that leads to the exception	Whenever there is an attempt to divide an integral or decimal by zero.
Who is in charge of throwing the exception	The programmer, should check before division whether the denominator is zero and throw the exception.
What details would you provide to the callers when throwing this exception	• The type of the exception • The values attempted to be divided • The method name
Can the exception be generally caught	Yes this can be caught and if the input is validated before division, the exception avoided.
Should you catch this exception type or pass to the user	Catch and handle where it relates to internal code, but otherwise passed to the user.
Is the exception a case when you want to avoid the exception to occur? If so, what would be your actions as a programmer to avoid it?	Yes, this can generally be avoided by guarding against a 0 denominator.

References

https://docs.microsoft.com/en-us/dotnet/api/system.dividebyzeroexception?view=netframework-4.8

https://www.dotnetperls.com/dividebyzeroexception

https://stackoverflow.com/questions/2601350/is-there-any-reason-to-throw-a-dividebyzeroexception

ArgumentException

Possible situation that leads to the exception	When arguments provided to a method are not valid.
Who is in charge of throwing the exception	The program should check the arguments and throw it where necessary. In general, it might be thrown by the CLR and indicate developer error.
What details would you provide to the callers when throwing this exception	• The type of the exception • The expected types of valid arguments • Details of the invalid argument
Can the exception be generally caught	Yes this can be generally caught and handled, although handling it may involve passing the exception to the caller. According to the C# documentation, one of the derived classes should be used instead of Argument Exception wherever possible, in order to provide more specific information. Alternatively a new class can be derived from Argument Exception to provide that additional detail by default.
Should you catch this exception type or pass to the user	Passed to the user where it is not possible to handle it gracefully without loss of information.
Is the exception a case when you want to avoid the exception to occur? If so, what would be your actions as a programmer to avoid it?	Yes, this is a case where avoiding the error is generally a good approach. Thorough and effective testing of code to the full extent of its intended use will assist to ensure that no Argument Exception can occur.

References

https://docs.microsoft.com/en-us/dotnet/api/system.argumentexception?view=netframework-4.8

https://stackoverflow.com/questions/774104/what-exceptions-should-be-thrown-for-invalid-or-unexpected-parameters-in-net

ArgumentOutOfRangeException

Possible situation that leads to the exception	As a derived class of ArgumentException, this is thrown when an argument to a method is outside the allowed range of values defined by the method.
Who is in charge of throwing the exception	The method that places the limit on the range of values.
What details would you provide to the callers when throwing this exception	• The type of the exception • The argument that is outside the range • Acceptable range for the argument
Can the exception be generally caught	Yes, this can be generally caught and handled. This can be specifically thrown after checking the arguments provided to a method call.
Should you catch this exception type or pass to the user	Catch and handle if possible. Otherwise pass to the caller.
Is the exception a case when you want to avoid the exception to occur? If so, what would be your actions as a programmer to avoid it?	Not necessarily. It may be perfectly acceptable for this exception to occur and to trigger a specific approach to handling it. For example, in Task 2.3C, this was used in the NextHour, NextMinute and NextSecond methods to know when the upper limit of the range had been exceeded and a new minute, hour or day was reached.

References

https://docs.microsoft.com/en-us/dotnet/api/system.argumentoutofrangeexception?view=netframework-4.8

http://www1.cs.columbia.edu/~lok/csharp/refdocs/System/types/ArgumentOutOfRangeException.html

https://stackoverflow.com/questions/9900481/system-argumentoutofrangeexception-argument-is-out-of-range-error-in-a-shortes

SystemException

Possible situation that leads to the exception	This class derives directly from Exception and forms the base class of the other exceptions investigated in this task. As such, the polymorphic behavior of the subclasses often indicates a subclass error, even when throwing a SystemException. However, in general this is reserved for the CLR and may be thrown when there is a system level exception, and allows differentiation of exceptions with with ApplicationException and classes derived from it.
Who is in charge of throwing the exception	Normally, the CLR. As an important note in the C# documentation: Because SystemException serves as the base class of a variety of exception types, your code should not throw a SystemException exception, nor should it handle a SystemException exception unless you intend to re-throw the original exception.
What details would you provide to the callers when throwing this exception	• As per the above, re-throw the original exception when not using a more specific, derived class and this exception occurs.
Can the exception be generally caught	Yes, however although my program specifically throws this exception, it was always a subclass that then was thrown and caught.
Should you catch this exception type or pass to the user	No. It should not be caught, unless the intend is to rethrow the original exception.
Is the exception a case when you want to avoid the exception to occur? If so, what would be your actions as a programmer to avoid it?	Ideally, yes but since this is not an exception normally being thrown by a programmer, it may be difficult to guard against, as it will be thrown primarily by an interruption elsewhere in the system, outside the program.

References

https://docs.microsoft.com/en-us/dotnet/api/system.systemexception?view=netframework-4.8

http://etutorials.org/Programming/programming+microsoft+visual+c+sharp+2005/Part+III+More+C+Language/Chapter+9+Exception+Handling/System.Exception/

```cs
using System;
using System.Text;

namespace Task_4._1P
{

    /// <summary>
    /// Helper class for the exception implementations. It models
    /// a simple account class for an account holder and their balance.
    /// </summary>
    class Account
    {
        public string FirstName { get; private set; }
        public string LastName { get; private set; }
        public int Balance { get; private set; }

        /// <summary>
        /// Constructor for an account
        /// </summary>
        /// <param name="firstName">Account holder's first name</param>
        /// <param name="lastName">Account holder's last name</param>
        /// <param name="balance">Balance of the account as an int</param>
        public Account(string firstName, string lastName, int balance)
        {
            FirstName = firstName;
            LastName = lastName;
            Balance = balance;
        }

        /// <summary>
        /// Attempts to withdraw funds from the account if sufficient
        /// funds are available
        /// </summary>
        /// <param name="amount">The amount to attempt to withdraw</param>
        /// <exception cref="System.InvalidOperationException">Thrown
        /// when the amount to withdraw is more than the available
        /// funds</exception>
        public void Withdraw(int amount)
        {
            if (amount > Balance)
            {
                throw new InvalidOperationException("Insufficient funds");
            }
            Balance = Balance - amount;
        }
    }

    /// <summary>
    /// Empty class definition to assist with the ArgumentException
    /// example
    /// </summary>
    public class MyClass { }

```

```
54      /// <summary>
55      /// Helper class for the ArgumentOutOfRangeException, which we
56      /// encountered in Task 2.3C
57      /// </summary>
58      class MyTime
59      {
60          // Instance variables
61          private int _hour;
62          private int _minute;
63          private int _second;
64
65          /// Reference for this approach, from:
66          /// https://stackoverflow.com/questions/56197825
67          public int Hour
68          {
69              get => _hour;
70              set => _hour = (value >= 0) && (value <= 23)
71                  ? value
72                  : throw new ArgumentOutOfRangeException("Invalid hour. Must be
                    ↪  0-23");
73          }
74
75          public int Minute
76          {
77              get => _minute;
78              set => _minute = (value >= 0) && (value <= 59)
79                  ? value
80                  : throw new ArgumentOutOfRangeException("Invalid minute. Must be
                    ↪  0-59");
81          }
82
83          public int Second
84          {
85              get => _second;
86              set => _second = (value >= 0) && (value <= 59)
87                  ? value
88                  : throw new ArgumentOutOfRangeException("Invalid second. Must be
                    ↪  0-59");
89          }
90
91          public MyTime(int hour, int minute, int second)
92          {
93              Hour = hour;
94              Minute = minute;
95              Second = second;
96          }
97      }
98
99      class Program
100     {
101         // Theoretically sets a limit on the number of recursive
102         // calls in the execute method, but this is bypassed in
103         // the implementation
```

```
104         const int MAX_RECURSIVE_CALLS = 1000;
105
106         /// <summary>
107         /// Helper method for the StackOverflowException
108         /// </summary>
109         /// <param name="counter"></param>
110         static void Execute(int counter)
111         {
112             counter++;
113
114             if (counter <= MAX_RECURSIVE_CALLS)
115                 counter--;
116
117             Execute(counter);
118         }
119
120         public static void Main(string[] args)
121         {
122             // NullReference Example
123             int[] values = null;
124
125             try
126             {
127                 // NullReferenceException occurs here because the
128                 // loop attempts to set a value within the array
129                 // but the size of the array has not been set
130                 // so no position can be addressed, as it is null
131                 for (int i = 0; i < 10; i++)
132                     values[i] = i * 2;
133
134                 foreach (var value in values)
135                     Console.WriteLine(value);
136             }
137             catch (NullReferenceException exception)
138             {
139                 Console.WriteLine("The following error detected: "
140                     + exception.GetType().ToString()
141                     + " with message \"" + exception.Message + "\"");
142             }
143
144             // IndexOutOfRange Example
145             try
146             {
147                 values = new int[10];
148
149                 // IndexOutOfRangeException occurs here because the loop
150                 // attempts to set a value for an index equal to the
151                 // length of the array, which is one more than the last
152                 // available index value (ie. values[10] is beyond the
153                 // end of the array)
154                 for (int i = 0; i <= values.Length; i++)
155                 {
156                     values[i] = i * 2;
```

```
157                 }
158
159                 foreach (var value in values)
160                     Console.WriteLine(value);
161
162             }
163             catch (IndexOutOfRangeException exception)
164             {
165                 Console.WriteLine("The following error detected: "
166                     + exception.GetType().ToString()
167                     + " with message \"" + exception.Message + "\"");
168             }
169
170             // StackOverflow Example
171             try
172             {
173                 // Will cause recursive filling of the stack with
174                 // increment and decrement steps
175                 Execute(0);
176             }
177             catch (StackOverflowException exception)
178             {
179                 // This catch block will never be executed as a
180                 // stack overflow always terminates the program
181                 Console.WriteLine("The following error detected: "
182                     + exception.GetType().ToString()
183                     + " with message \"" + exception.Message + "\"");
184             }
185
186             // OutOfMemory Example
187             try
188             {
189                 // OutOfMemory occurs because the capacity and length
190                 // are set smaller than the amount of memory required
191                 // to create the initial string and then insert
192                 // the second string into the first at index 0
193                 StringBuilder sb = new StringBuilder(15, 15);
194                 sb.Append("Substring #1 ");
195                 sb.Insert(0, "Substring #2 ", 1);
196             }
197             catch (OutOfMemoryException exception)
198             {
199                 Console.WriteLine("The following error detected: "
200                     + exception.GetType().ToString()
201                     + " with message \"" + exception.Message + "\"");
202             }
203
204             // InvalidCast Example
205             try
206             {
207                 bool flag = true;
208                 char ch = Convert.ToChar(flag); // bool cannot cast to char
209             }
```

```
210             catch (InvalidCastException exception)
211             {
212                 Console.WriteLine("The following error detected: "
213                     + exception.GetType().ToString()
214                     + " with message \"" + exception.Message + "\"");
215             }
216 
217             // DivideByZeroException Example
218             try
219             {
220                 int x = 1000;
221                 int y = 0;
222 
223                 // This is a simple and explicitly coded 0, however this
224                 // is more relevant where a method involves some division
225                 // involving arguments and/or where user or sensor input
226                 // is involved
227                 Console.WriteLine(x / y);
228             }
229             catch (DivideByZeroException exception)
230             {
231                 Console.WriteLine("The following error detected: "
232                     + exception.GetType().ToString()
233                     + " with message \"" + exception.Message + "\"");
234             }
235 
236             // ArgumentException Example
237             try
238             {
239                 MyClass my = new MyClass();
240                 string s = "test text";
241                 int i = s.CompareTo(my); // comparing object to string
242             }
243             catch (ArgumentException exception)
244             {
245                 Console.WriteLine("The following error detected: "
246                     + exception.GetType().ToString()
247                     + " with message \"" + exception.Message + "\"");
248             }
249 
250             // ArgumentOutOfRangeException Example
251             try
252             {
253                 MyTime t = new MyTime(24, 0, 0); // 24 is larger than the allowed
                    ↪ range
254             }
255             catch (ArgumentOutOfRangeException exception)
256             {
257                 Console.WriteLine("The following error detected: "
258                     + exception.GetType().ToString()
259                     + " with message \"" + exception.Message + "\"");
260             }
261 
```

```
262             // SystemException Example
263             try
264             {
265                 // This is the base class for other exceptions and
266                 // is normally reserved for runtime errors in the
267                 // CLR and a more specific exception type should be derived
268                 // or thrown
269                 int[] array = new int[5];
270                 array[10] = 25; // This is also an IndexOutOfRangeException
271             }
272             catch (SystemException exception)  // This is normally bad
273             {
274                 Console.WriteLine("The following error detected: "
275                     + exception.GetType().ToString()
276                     + " with message \"" + exception.Message + "\"");
277             }
278 
279             // Original code supplied in the task for an
280             // InvalidOperationException
281             try
282             {
283                 Account account = new Account("Sergey", "P", 100);
284                 account.Withdraw(1000);
285             }
286             catch (InvalidOperationException exception)
287             {
288                 Console.WriteLine("The following error detected: "
289                     + exception.GetType().ToString()
290                     + " with message \"" + exception.Message + "\"");
291             }
292         }
293     }
294 }
```