Task 4.1

Exception Handling Report

Student ID – DFCS|DK|62|203

Table of Content

What is Exception Handling

The technique of managing undesirable or unexpected events that arise during the execution of a computer program is known as exception handling. In the absence of exception handling, exceptions would impede a program's normal operation. The occurrence are handled by exception handling to keep the program or system from c crashing.

A number of things can result in exceptions, including improper user input, Programming mistakes, device malfunctions, lost network connections, memory conflicts with other apps, attempts by programs to divide by zero, and users attempting to access unavailable files.

When an exception arises in the middle of a multi-statement program, the program crashes because the statements that com after the exception are not executed. Exception handling assists in ensuring that an exception doesn't occur.

Exception handling allows exceptions to be throw and caught. If a detecting function in a block of code is unable to handle an anomaly, an exception is sent to a function that can handle it. A catch statement is a group of lines that handle a specific thrown exception. The catch parameters determine the specific type of exception that is thrown.

Exception handling is useful for handling exceptions that are not manageable locally. Rather that showing an error massage in the program the exception handler shifts control to the area where the error can be addressed. Whether to handle or throw exceptions is up to the function.

Another technique to tell error-handling code from normal code is to look for try blocks, or code enclosed in curly braces or brackets that could raise an exception. Try blocks allow programmers to more easily manage exception objects.

Let's now examine a few exceptions add their methods for addressing them.

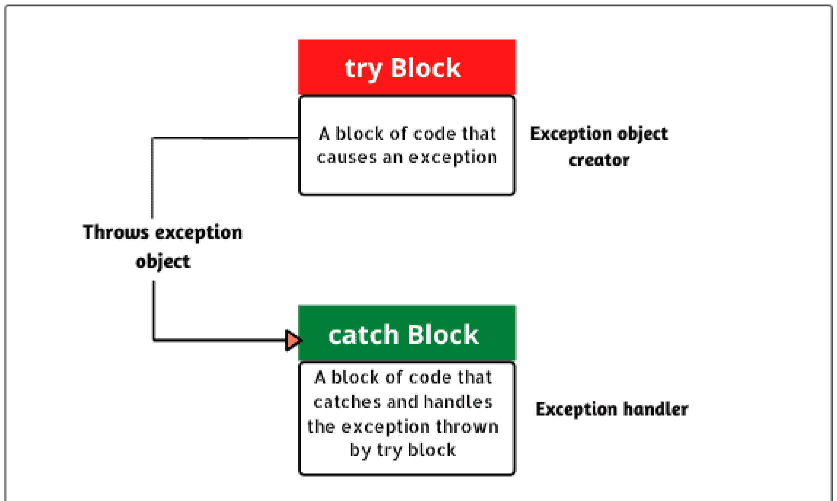

try Block
A block of code that causes an exception
Exception object creator
Throws exception object
catch Block
A block of code that catches and handles the exception thrown by try block
Exception handler

Fig: Exception handling mechanism

NullReferenceException

- You get an exception called NullReferenceException when you attempt to use a member of a type whose value is null.
- The runtime system raises a NullReferenceException when you try to access a member (property or function) of a null reference. Programmers may unintentionally contribute to it by handling potentially null values from outside sources or by initializing variables incorrectly. Programmers generally want to try to stay away from throwing this exception. Instead, null tests ought to be run prior to object access.
- The parameter and message are:
    - ✓ Depending on the circumstances, the exact message may vary, but in general, it indicates that an attempt was made to contact a member of a null reference. A ".NET Framework" instance may display "Object reference not set to an object."
    - ✓ Occasionally, we are able to add as parameters the kind of null reference and the member that was attempted to be accessed.
- Indeed, a try-catch block can be used to catch and handle NullReferenceExceptions.
- Although it is technically possible, we shouldn't catch a NullReferenceException because doing so regularly has more disadvantages than advantages:
    - ✓ Masking underlying issues: Managing and identifying the exception masks the true source of the problem, which makes debugging and fixing it more challenging.
    - ✓ Encourages shoddy coding techniques when writing safe, well-structured code, relying too much on catching NullReferenceExceptions might lead to complacency.
    - ✓ NullReferenceExceptions are challenging to handle correctly because they typically indicate logical errors in the program's architecture.
- Programmers should always ensure that an object is null before attempting to use its methods or attributes. The Null Reference Exception is one of the more common problems, albeit not a major one. Verifying the variable or property before accessing it is one easy way to stop it, and testing the variable inside of an if statement is a pretty easy way to do this.

IndexoutOfRangeException

- The exception that is raised when a member of an array or collection is attempted to be accessed using an index that is outside of its boundaries
- This exception is often raised by the runtime environment. It starts this procedure right away when it notices an attempt to access an array or collection element that has an invalid index. Theoretically, we shouldn't manually throw this exception as programmers. It is meant to be thrown by the runtime system to signal a specific error state. Throwing it yourself could lead to strange behavior and make code maintenance more challenging.
- The parameter and message
  - ✓ Depending on the code, the message would alter. For instance, I could provide an exception that looked something like this: "System.IndexOutOfRangeException: Index has to be >= lower bound and \= upper bound of the array."
  - ✓ The name or type of the collection as well as the invalid index may be included in the exception constructor.
- Sure, a try-catch block can be used to catch and manage it.
- Yes, it can be caught, but developer errors usually result in an IndexOutOfRangeException exception being produced. We should identify the problem's root cause and make the necessary code corrections rather than managing the exception. However, handling is still useful for trying to do recovery activities or for giving precise error messages. However, at the same time, take care not to hide basic logical errors.
- Few avoidance techniques exist.
  - ✓ Comprehensive validation: Verify user input or computed indices before accessing array elements.
  - ✓ Use loop constructs: For each loops are a better method for iterating across arrays than manual indexing because they implicitly handle bounds checking.
  - ✓ Length property: To determine the range of allowed indices for an array, use the Length property.

StackeOverflowException

- The exception that is raised when the size of the stack is exceeded by the execution stack. It may occur from a method that keeps calling itself without adding a proper breakpoint, which can cause StackOverFlowExeption.

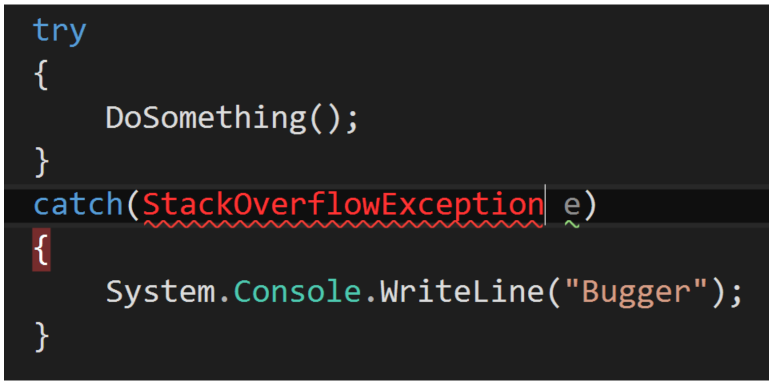
- Theoretically, a programmer may write code that contains an infinite recursive loop and purposefully throw a `StackOverflowException}. However, as it is a system-level exception that notifies the runtime environment of a critical condition, it is advised against throwing it on your own. Should you attempt to throw it yourself, you can run into issues with debugging, strange behavior, and even crashes.

- The parameters and message:
  - ✓ The user would be informed that "an infinite recursion has occurred, causing the stack to overflow."
  - ✓ Normally, I wouldn't supply the parameter, but if I did, I would provide details about the code that was causing the problem.

- Cannot be caught in a traditional try-catch block in.NET (since.NET 2.0). When it tries to catch it, the software usually crashes because it was terminated. As a result, it cannot be managed as effectively.

- Practically speaking, this exception is typically uncatchable. Rather
  - ✓ Put more emphasis on prevention than handling.
  - ✓ Rewrite the code to minimize overuse of recursion or use different strategies.

- Yes, this exception would cause a serious software crash, thus generally speaking, we should always attempt to avoid it in our applications. To steer clear of it:
  - ✓ To limit the depth of recursion, limit the number of calls or, if possible, employ iterative techniques.
  - ✓ Utilize tail recursion optimization in specific circumstances to avoid stack growth (assuming the language permits it).
  - ✓ Non-recursive techniques like memory management, stacks, and loops can be used to solve problems without resorting to recursion.



```
try
{
    DoSomething();
}
catch(StackOverflowException e)
{
    System.Console.WriteLine("Bugger");
}
```

OutOfMemoryException

- The error that appears when a program's memory cannot be used to continue its execution.

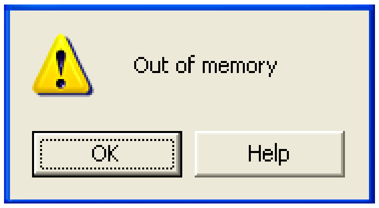
- The programmer does not throw OutOfMemoryException; the runtime system does. This is a system-level exception that alerts the runtime environment to a critical situation. Throwing it by hand may result in erratic behavior and even crashes.

- Although throwing OutOfMemoryException directly is not advised, in the unlikely event that we did, I would:
  - ✓ Indicate clearly that there has been an out-of-memory error. For instance, "The application has run out of available memory." To free up memory, please consider restarting your computer or shutting off other apps.
  - ✓ The parameters would be the code that is creating the problem, the requested and available memory amounts.

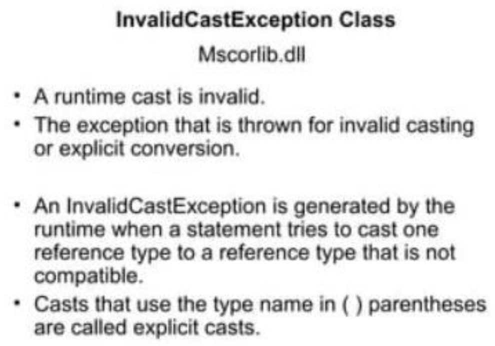
- A try-catch block can be used to manage and catch it.

- This type of OutOfMemoryException exception indicates a catastrophic failure. Add a catch block to your code that invokes the Environment. If you choose to handle the exception, use FailFast. To terminate your application and write a note to the system event log, use the FailFast technique.

- Yes, in order to prevent any software crashes, it must be avoided. Some preventive measures include:
  - ✓ Write code that efficiently utilizes memory, stays away from pointless allocations, and promptly releases resources when it's finished.
  - ✓ Keep an eye out for potential bottlenecks in memory usage during testing and development so that preventive measures can be taken.
  - ✓ If memory constraints are a persistent issue, look into memory-saving techniques like caching, compression, or memory-mapped files.


Out of memory

InvalidCastException


InvalidCastException Class

Mscorlib.dll

- A runtime cast is invalid.
- The exception that is thrown for invalid casting or explicit conversion.
- An InvalidCastException is generated by the runtime when a statement tries to cast one reference type to a reference type that is not compatible.
- Casts that use the type name in ( ) parentheses are called explicit casts.

- When casting something incorrectly or converting something explicitly, an exception is raised.

- You as a programmer and the runtime system can both throw InvalidCastExceptions.
  - ✓ Runtime system: This exception is automatically thrown when it identifies an invalid attempt to convert a value from one type to another at runtime.
  - ✓ Programmer: You may also explicitly throw something with the throw command to show that a conversion in your code logic is erroneous.

- Indicates that an attempt was made to convert between incompatible types; specify the type that was anticipated as well as the type that was actually encountered. "Invalid cast from type'string' to type 'int'," for instance. 'Hello' cannot be expressed as an integer.

- Indeed, you may typically use a try-catch block in your code to capture and handle InvalidCastExceptions.

- Two instances exist, one for catching exceptions and the other for giving them to users;
  - ✓ Catching: When thinking about catching, be sure to put recovery methods in place, such as trying again with updated input or giving default values.
  - ✓ If you're not sure how to handle an invalid cast politely, you might want to think about handing it to the user. Doing so can help you avoid unanticipated downstream issues. This enables the user or higher-level programming to determine the best course of action.

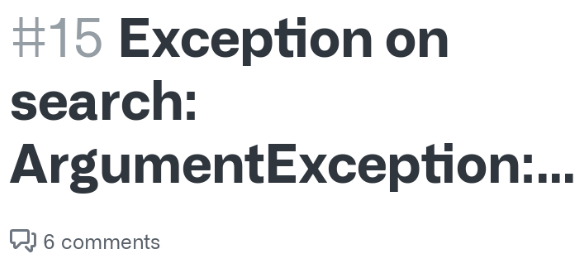
- It is generally preferable to avoid InvalidCastException in your program. While it occasionally can be useful for revealing type errors and maintaining type safety, relying too much on it can highlight potential issues with the design or type handling of your code.

DivideByZeroException

- The exception raised whenever a decimal or integral value is attempted to be divided by zero.
- DivideByZeroException is thrown by the runtime system. Although it is a system-level exception that indicates a serious mathematical error and the runtime system is built to handle it properly, it can theoretically be thrown manually by a programmer. When a programmer attempts to divide an integer by zero, it may generate an error. Throwing this exception on purpose is uncommon, though, as it typically indicates a mistake.
- It is generally not advised to manually throw this exception; if we were to do so, we could theoretically specify a message and parameter;
  - ✓ Message: State clearly that an attempt has been made to divide by zero. As an illustration, "Error: Division by zero is not permitted." Kindly make sure the values you are dividing are correct."
  - ✓ Parameter: To help with debugging, include the values that were divided. Give some background on the section of the code where the division was attempted, if at all feasible. For instance, "I tried using the 'CalculateAverage' function to divide 50 by 0." Kindly check the input values."
- In general, a try-catch block can be used to catch and handle the {DivideByZeroException}.
- Generally speaking, it's preferable to let the runtime system handle the exception rather than trying to catch it yourself.
  This is a basic mathematical issue that the runtime is designed to handle correctly: DivideByZeroException. Its internal processes could be upset by manually catching it, which could result in unpredictable behavior. For the purpose of debugging and troubleshooting, the runtime precisely determines the point of division by zero.
- Absolutely, you should try to avoid using DivideByZeroException in your application. As a programmer, you can take the following steps to stop DivideByZeroException.
  - ✓ Input Validation: Carefully examine all possible zero divisors before dividing anything. Verify that the intermediate results, data from outside sources, and user input are all non-zero.
  - ✓ Conditional Logic: When the divisor might be zero, use if statements or conditional expressions to completely avoid division.
  - ✓ Extensive Testing: Create unit tests that cover a wide range of input cases, including those involving probable zero divisors.

ArgumentException

- The exception that is raised when a method receives an invalid argument.
- Programmers like you have the ability to throw ArgumentExceptions, as does the runtime system.
  - ✓ Runtime system: This exception is automatically thrown when a method's argument doesn't match the required parameters.
  - ✓ Coder: An incorrect argument received in your code can also be signaled clearly by using the toss command.
- Message: Indicate that an invalid argument was supplied to a method, which is the reason for the error. Mention the intended argument's format, range, and type. "ArgumentException: 'invalidName' is not a valid username," for instance. Usernames must consist of five to twenty characters and be alphanumeric."
- Parameter: Indicate which parameter was given the incorrect value. "ArgumentException: 'name' parameter at line 25" is one example. Got 'invalidName' when I was expecting an alphanumeric string with five to twenty characters.
- Indeed, a try-catch block can be used to catch and handle the {ArgumentException} in most cases.
- Whether to catch or pass an ArgumentException relies on a number of variables in your particular situation.
  - ✓ When context-specific recovery, enlightening messaging, and enhanced user experience are critical, catch ArgumentExceptions.
  - ✓ When modularity, accountability, and consistency are valued and the caller is capable of managing them, pass ArgumentExceptions to the caller.
  - ✓ To sum up, whenever the {ArgumentException} arises, it's usually preferable to catch and manage it within your code. Taking care of it enables you to deliver informative error messages and guarantee that invalid arguments don't result in unexpected behavior or crashes.
- Generally speaking, you should steer clear of ArgumentException in your program. Several tactics for avoiding:
  - ✓ Descriptive parameter names: Give each argument a name that explains its expected use and limitations.
  - ✓ Verify user input: Before sending data to methods in user interfaces, thoroughly validate the data to make sure it follows expected formats and ranges.



#15 **Exception on search: ArgumentException:...**

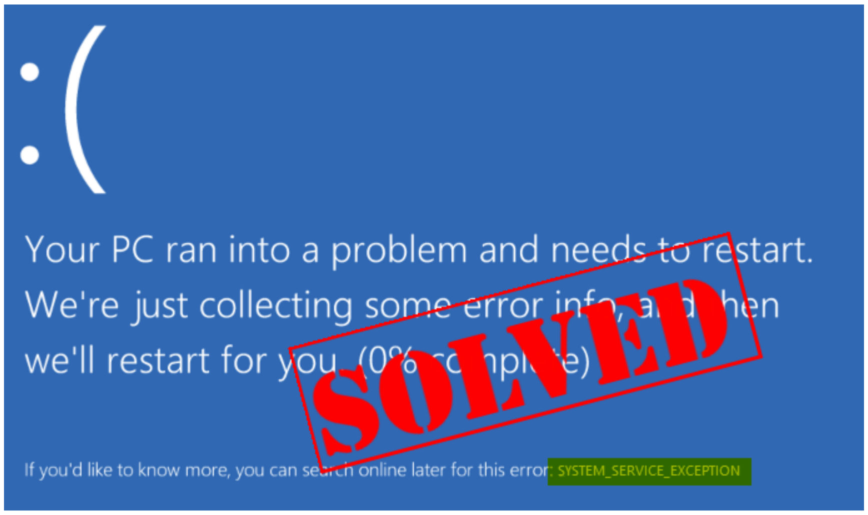
6 comments

ArgumentOutOfRangeException

- The exception that is raised when a parameter's value is beyond of the permitted range specified by the invoked method.
- ArgumentOutofRangeException can be thrown by the runtime system as well as by you as a programmer.
    - ✓ If a method or function's anticipated range is outside of an incorrect parameter value, you can explicitly issue ArgumentOutOfRangeException by using the throw statement.
    - ✓ Yes, as a coder, throwing ArgumentOutOfRangeException is frequently a smart idea. It accomplishes important goals such as early error detection and upholding the legitimacy of arguments.
      Refactor code, if at all possible, to prevent scenarios in which out-of-range parameters might be passed
- Message: Indicate that an argument value that is outside of the anticipated range is the cause of the exception. Indicate in detail the argument's acceptable range or range of values. "ArgumentOutOfRangeException: The argument 'age' must be between 18 and 120," for instance, is one example. A value of '5' was supplied.
- Parameter: Indicate which parameter was given the incorrect value. "ArgumentOutOfRangeException: Parameter 'index' at line 32," for instance. '5' is an invalid value. It has to be less than the array's size and nonnegative (4)."
- The decision to pass or catch depends on the circumstances:
    - ✓ Catch when: It's critical to have context-specific recovery, educational messaging, and enhanced UX.
    - ✓ When to pass: Priorities include flexibility, accountability clarity, and consistent behavior. Caller is able to manage them well
    - ✓ Generally speaking, it is preferable to catch and handle the {ArgumentOutOfRangeException} within your code. By handling it, you can avoid the usage of erroneous arguments, which could result in unexpected behavior or errors, and give a clear error notice.

SystemException

- Acts as the namespace's foundational class for system exceptions. The purpose of this class is to help distinguish between application and system exceptions. This exception class serves as the foundation for others like ArgumentException, FormatException, and InvalidOperationException.
- SystemException is thrown by the runtime system, not by you as a programmer. The common language runtime (CLR) handles internal exceptions for unexpected and serious mistakes that arise within the system or within itself. In theory, you ought to stay away from it in general. This is the reason why:
  - ✓ Throwing a SystemException might make it more difficult to diagnose and resolve the issue by hiding the real source of the mistake.
  - ✓ Throwing a SystemException could interfere with the CLR's core operations, causing unpredictable behavior and even the program to fail.
- Although throwing SystemException directly is not advised, in the event that you do so due to extraordinary circumstances;
  - ✓ Message: Using language that the intended audience can comprehend, describe the issue that arose with the runtime or underlying system.
  - ✓ Parameter: Specify the precise kind of SystemException (such OutOfMemoryException or ExecutionEngineException) that was thrown.
- Yes, in general, you want to keep your program from throwing SystemExceptions. As the runtime system throws it, you have no direct control over when it occurs, but you can take proactive steps to reduce the possibility of circumstances that could lead to it:
  - ✓ Early Correction of Possible Errors
  - ✓ Test Your Code Often
  - ✓ Keep an eye out for system exceptions
  - ✓ Maintain Updating Your Runtime Environment
  - ✓ Utilize Sturdy Coding Methods



Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you. (0% complete)

If you'd like to know more, you can search online later for this error: SYSTEM_SERVICE_EXCEPTION

Reference

- More, S. (2023,April 15). C# *NullReferenceException Class (System)*. Microsoft Learn.
- *Newest "stack-overflow" questions*. (n.d.). Stack Overflow.
- Pedamkar, P . (2023, April 13). *C# OutOfMemoryException*. EDUCBA.
- *C# InvalidCastException – Dot Net Perls*. (n.d).
- *Dotnet-Bot. (a.n.d.-a). DividebyZeroException Class (System). Microsoft Learn.*
- *Dotnet-Bot. (a.n.d.-a). ArgumentException class (System). Microsoft*
- *Dotnet-bot. (a.n.d.-b). ArgumentOutOfRangeexception Class (system). Microsoft Learn.*
- Dotnet-Bot. (a.n.d.-i). System Exception class (System). Microsoft Learn