Task 4.1P

REPORT ON EXCEPTION HANDLING

Lucifer Morningstar

SIT232 Object Oriented Programming

Table of contents

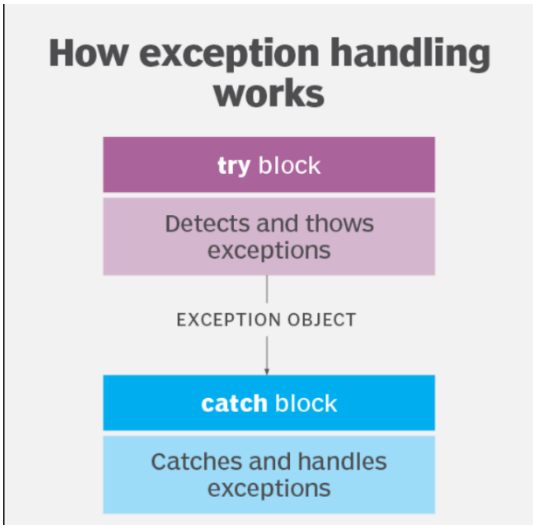
What is Exception Handling

When a computer program executes, handling exceptions is the process of handling undesired or unexpected events. Without exception handling, exceptions would interfere with a program's regular functioning. Exception handling handles these events to prevent the application or system from crashing.

Exceptions can arise from a variety of causes, such as incorrect user input, programming errors, device malfunctions, lost network connections, memory conflicts with other applications, programs trying to divide by zero, or users trying to access files that aren't accessible.

In the event that an exception occurs mid-execution in a program with many statements, the program crashes because the statements that follow the exception do not run. When an exception arises, exception handling helps make sure that this doesn't happen.

Exceptions can be thrown and caught by exception handling. An exception is delivered to a function that can manage it if a detecting function in a block of code is unable to handle an abnormality. A collection of lines that deal with a particular thrown exception is called a catch statement. The particular kind of exception that is thrown depends on the catch parameters.



When addressing exceptions that cannot be handled locally, exception handling comes in handy. The exception handler moves control to the area where the error can be handled, rather than displaying an error status in the application. A function has the option of handling exceptions or throwing them.

Try blocks, or code encased in curly braces or brackets that could create an exception, are another way to distinguish error-handling code from regular code. Programmers can organize exception objects more easily by using try blocks.

Now let’s look at some of the exceptions and how they work on handling the exception.

NullReferenceException

1. A NullReferenceException exception is thrown when you try to access a member on a type whose value is null.

2. When you attempt to access a member (property or method) of a null reference, the runtime system throws a NullReferenceException. Programmers might indirectly cause it by handling potentially null values obtained from external sources or by improperly initializing variables. Generally speaking, as a programmer, you should try to avoid throwing this exception. Null tests should be carried out instead before accessing objects.

3. The message and parameter:

   - The precise message might change based on the situation, but generally speaking, it makes it clear that an attempt was made to access a member of a null reference. An instance of ".NET Framework" could show "Object reference not set to an object."

   - We can sometimes include the type of the null reference and the member attempted to be accessed as parameters.

4. Yes, you can catch a NullReferenceException and handle it using a try-catch block.

5. We should not catch a NullReferenceException although theoretically feasible, capturing it frequently has more drawbacks than benfits:

   - Masking underlying problems: Handling and catching the exception obscures the real cause of the issue, making troubleshooting and correction more difficult.
   - Promotes poor coding practices: Writing secure, well-organized code might become complacent when one depends too much on catching NullReferenceExceptions.
   - Difficult to handle effectively: NullReferenceExceptions are tricky to handle correctly because they frequently point to logical mistakes in the program's architecture.

6. Before attempting to access an object's attributes or methods, programmers should always make sure that the object is null. Although not a serious issue, the Null Reference Exception is one of the more frequent ones. One straightforward method to prevent it is to verify the variable or property before accessing it and checking the variable inside of an if statement is a really simple method to accomplish this.

IndexOutOfRangeExeption

1. The exception that is thrown when an attempt is made to access an element of an array or collection with an index that is outside its bounds.

2. Generally, this exception is thrown by the runtime system. When it detects an attempt to access an array or collection element with an invalid index, it immediately initiates this process. In theory, as a programmer, we shouldn't throw this exception manually. It is intended for the runtime system to throw it in order to indicate a particular error state. Throwing it yourself can result in unexpected behavior and increase the difficulty of maintaining the code.

3. The message and parameter:

    - The message would change according to the code, for example I would provide an exception like this: "System.IndexOutOfRangeException: Index has to be >= lower bound and <= upper bound of the array".
    - The exception constructor might include the invalid index and the collection's name or type.

4. Yes, it can be caught and handled using a try-catch block.

5. Yes, it can be caught but typically, an IndexOutOfRangeException exception is thrown as a result of developer error. Instead of handling the exception, we should diagnose the cause of the error and correct our code. but still handling can be beneficial for providing specific error messages or attempting recovery actions. But at the same time make sure not to cover up fundamental logical mistakes.

6. There are few avoidance strategies:

    - Extensive validation: Prior to accessing array elements, validate user input or computed indices.
    - Employ loop constructs: Foreach loops, which take care of bounds checking implicitly, are a better option for iterating through arrays than manual indexing.
    - Length property: Use the Length property to find the range of indices that are acceptable for an array.

StackOverflowException

1. The exception that is thrown when the execution stack exceeds the stack size. It can result from a method continually calling itself without a suitable termination condition, which can lead to StackOverFlowExeption.

2. It is theoretically possible for a programmer to intentionally throw a `StackOverflowException` by writing code that incorporates an infinite recursive loop. But it is recommended to not throw it by yourself because it is a system-level exception that alerts the runtime environment to a critical situation. If you try to throw it yourself, you may encounter debugging difficulties, erratic behavior, and possible crashes.

3. The Message and parameters:
   - The message to user would be “the stack has overflowed due to infinite recursion has occurred”.
   - The parameter would be not given generally but if I want to give, I would give the information about the code responsible for the issue.

4. Cannot be captured in.NET (since.NET 2.0) using a conventional try-catch block. Usually, the program will be terminated when it tries to catch it causing for a program crash. Therefore, it cannot be handled as well

5. Generally, this exception cannot be caught when practically speaking. Instead:

   - Focus on prevention rather than handling.
   - Redesign code to avoid excessive recursion or implement alternative approaches.

6. Yes, in general we should always try to avoid this exception in our application because it would lead to a critical program crash. In order to avoid it:
   - Set a maximum number of calls or, if practical, use iterative methods to reduce the depth of recursion.
   - In certain situations, use tail recursion optimization (if the language allows it) to prevent stack expansion.
   - To address issues without using recursion, take into consideration non-recursive methods like memoization, stacks, and loops.

OutOfMemoryException

1. The exception that is thrown when there is not enough memory to continue the execution of a program.

2. The runtime system throws OutOfMemoryException, not the programmer. It's a system-level exception that signals a critical condition within the runtime environment. Manually throwing it can lead to unpredictable behavior and potential crashes.

3. While it's not recommended to throw OutOfMemoryException directly, if we were to do so in a hypothetical scenario, I would:

    - Clearly state that an out-of-memory error has occurred, for example, “The application has run out of available memory. Please try closing other programs or restarting your computer to free up memory.”.
    - The parameters would include, amount of memory requested, available memory, and the code that is causing the issue.

4. It can be caught and handled using a try-catch block.

5. An OutOfMemoryException exception of this kind denotes a disastrous failure. Include a catch block that calls the Environment.FailFast if you decide to handle the exception. Use the FailFast method, to end your application and add a record to the system event log.

6. Yes, it needs to be avoided to prevent any program crashes, some preventive strategies are;

    - Write code that makes effective use of memory, avoids needless allocations, and releases resources quickly when they're done.
    - Monitor memory use in testing and development to spot possible bottlenecks and take preventative action.
    - Investigate memory-saving strategies like caching, compression, or memory-mapped files if memory limitations are a recurring problem.

InvalidCastException

1. The exception that is thrown for invalid casting or explicit conversion.

2. Both the runtime system and you as a programmer can throw InvalidCastException.

   - Runtime system: When it detects an invalid attempt to convert a value during runtime from one type to another, it automatically throws this exception.
   - Programmer: To indicate an incorrect conversion in your code logic, you can also explicitly throw it using the throw command.

3.
   - The message can be given as indicating a conversion between incompatible types was attempted, mention both the type that was expected and the type that was actually encountered. For example, “Invalid cast from type 'string' to type 'int'. The value 'hello' cannot be converted to an integer.”
   - We can include the value that was being cast, as it can aid in debugging. For example, “Failed to convert the value '123abc' in variable 'userInput' to type 'int' in function 'CalculateTotal'.”

4. Yes, InvalidCastException can generally be caught and handled in your code using a try-catch block.

5. There are two instances, one for the catching the exception and other one passing it to the user;

   - Catching: when considering catching, make sure to implement recovery mechanisms like retrying with corrected input or providing default values.
   - Passing to user: consider passing to user if you're unsure how to handle the invalid cast gracefully, passing it to the user can prevent unexpected downstream errors. This allows higher-level code or the user to decide appropriate action.

6. Indeed, avoiding InvalidCastException in your program is generally desirable. Relying too much on it can reveal possible problems with the design or type handling of your code, even while it can occasionally be helpful for indicating type errors and upholding type safety.

   Some key avoidance strategies:
   - Thorough Type Checking: Employ compile-time type checking features like static typing and explicit type declarations to catch type mismatches early.
   - Runtime Validation: For dynamic scenarios where types aren't known beforehand, implement runtime type checks using is, as, or reflection.
   - Design for Type Safety:
     - Structure your code to minimize the need for casting by: Using generics to create type-safe collections and algorithms. Favoring polymorphism (interfaces and inheritance) over explicit casting when possible.

DivideByZeroException

1. The exception that is thrown when there is an attempt to divide an integral or Decimal value by zero.

2. The runtime system throws DivideByZeroException. But in theory it can be thrown manually by programmer even though It's a system-level exception that signals a critical mathematical error, and the runtime system is designed to handle it appropriately. A programmer can throw it when they try to divide a number by zero. However, it's not common to throw this exception deliberately because it usually represents an error.

3. Generally, it is not recommended to throw this exception manually, hypothetically we can provide a message and parameter if we were to do so;
   - Message: Explicitly mention that a division by zero has been attempted. For example, "Error: Division by zero is not allowed. Please check the values you are trying to divide."
   - Parameter: Include the values that were being divided, as this can aid in debugging. If possible, provide context about where in the code the division was attempted. For example, "Attempted to divide 50 by 0 in the function 'CalculateAverage'. Please verify the input values."

4. Yes, the `DivideByZeroException` can be generally caught and handled using a try-catch block.

5. it's generally better to let the runtime system handle the exception instead of catching it yourself. DivideByZeroException is a fundamental mathematical error, and the runtime is equipped to handle it appropriately. Manually catching it might disrupt its internal process and lead to unpredictable behavior.

   The runtime accurately detects the exact point of division by zero, providing precise information for debugging and troubleshooting.

6. Yes, it's generally preferable to avoid DivideByZeroException in your application. Here are actions you can take as a programmer to prevent DivideByZeroException.

   - Input Validation: Before performing any division, rigorously check for potential zero divisors. Validate user input, data from external sources, and intermediate results to ensure non-zero values.

   - Conditional Logic: Use if statements or conditional expressions to avoid division altogether when the divisor might be zero.

   - Thorough Testing: Write comprehensive unit tests to cover various input scenarios, including potential zero divisors.

ArgumentException

1. The exception that is thrown when one of the arguments provided to a method is not valid.

2. Both the runtime system and you as a programmer can throw ArgumentException.

   - Runtime system: It throws this exception automatically when an argument passed to a method doesn't meet the expected specifications.
   - Programmer: You can also throw it explicitly using the throw statement to signal an invalid argument received in your code.
   - Yes, you should throw ArgumentException in theory as a programmer.
     - Ensures methods receive intended types and values, promoting type safety and preventing unexpected behavior.
     - Catches invalid arguments early in the program flow, aiding debugging and troubleshooting.

3. Message: Specify that the exception is due to an incorrect argument passed to a method. Mention the expected type, range, or format of the argument. for example, "ArgumentException: 'invalidName' is not a valid username. Usernames must be alphanumeric and between 5 and 20 characters long."

   Parameter: Include the name of the parameter that received the invalid value. for example, "ArgumentException: 'name' parameter at line 25. Received 'invalidName' but expected an alphanumeric string between 5 and 20 characters".

4. Yes, the `ArgumentException` can be generally caught and handled using a try-catch block.

5. The decision to catch or pass ArgumentException depends on various factors in your specific context.
   - Catch ArgumentExceptions when context-specific recovery, informative messages, and improved user experience are crucial.
   - Pass ArgumentExceptions to the caller when modularity, clear responsibility, and predictable behavior are prioritized, and the caller can handle them effectively.
   - In conclusion, it's generally better to catch and handle the `ArgumentException` when it occurs within your code. Handling it allows you to provide meaningful error messages and ensure that invalid arguments do not cause unexpected behavior or crashes.

6. Yes, generally you want to avoid ArgumentException in your application. Some avoidance strategies:
   - Descriptive parameter names: Use names that clearly explain the expected purpose and constraints of each argument.
   - Validate user input: In user interfaces, perform thorough validation before passing data to methods to ensure it adheres to expected formats and ranges.

ArgumentOutOfRangeException

1. The exception that is thrown when the value of an argument is outside the allowable range of values as defined by the invoked method.
2. Both you as a programmer and the runtime system can throw ArgumentOutOfRangeException.

    - The runtime system automatically throws this exception when a method accesses an element of an array or collection using an index that is outside of the valid range.
    - You can explicitly throw ArgumentOutOfRangeException using the throw statement to signal an invalid argument value that falls outside of the expected range for a method or function.

    - Yes, it's often a good practice to throw ArgumentOutOfRangeException as a programmer. It serves valuable purposes, including: Enforcing argument validity, Early error detection. However, if possible, refactor code to avoid situations where out-of-range arguments are likely to be passed.

3. Message: Specify that the exception is due to an argument value that falls outside of the expected range. Clearly mention the valid range or set of values for the argument. for example, "ArgumentOutOfRangeException: The argument 'age' must be between 18 and 120. The value '5' was provided"

   Parameter: Include the name of the parameter that received the invalid value. for example, "ArgumentOutOfRangeException: Parameter 'index' at line 32. Value '5' is out of range. Must be non-negative and less than the size of the array (4)."

4. Yes, the `ArgumentOutOfRangeException` can be generally caught and handled using a try-catch block.
5. It depends on the situation whether to catch or pass:

    - Catch when: Context-specific recovery, informative messages, and improved UX are crucial.
    - Pass when: Modularity, clear responsibility, predictable behavior are prioritized. Caller can handle them effectively.
    - It's generally better to catch and handle the `ArgumentOutOfRangeException` when it occurs within your code. Handling it allows you to provide a clear error message and prevent the use of invalid arguments, which could lead to unexpected behavior or errors.

6. Yes, it's generally preferable to avoid ArgumentOutOfRangeException in your application. Some avoidance strategies:
    - Validate all user inputs thoroughly before passing them to methods.
    - Employ descriptive parameter names that clearly convey expected ranges.
    - Implement guard clauses at the beginning of methods to check argument validity early.
    - Use assertions within your code to verify assumptions about argument values during development.

SystemException

1. Serves as the base class for system exceptions namespace. This class is provided as a means to differentiate between system exceptions and application exceptions. It is the base class of such exceptions as ArgumentException, FormatException, and InvalidOperationException.

2. The runtime system, not you as a programmer, throws SystemException. It's an internal exception class handled by the common language runtime (CLR) for unexpected and severe errors that occur within itself or the underlying system. Theoretically you should generally avoid it.
   Here's why:
   - Throwing SystemException could disrupt the internal workings of the CLR, leading to unpredictable behavior and potentially crashing the program.
   - Throwing SystemException can obscure the actual cause of the error, making it difficult to debug and fix the problem.

3. While it's not recommended for you to throw SystemException directly, if you were to do so for exceptional circumstances;
   - message: Describe the problem that occurred within the runtime or underlying system, using language understandable to the target audience.
   - Parameter: Include the exact type of SystemException that was thrown (e.g., ExecutionEngineException, OutOfMemoryException).

4. No, you should generally not attempt to catch SystemException in your code.

5. It's generally not recommended to catch SystemException in your code. Here are the key reasons:
   - Unpredictable behavior: SystemException indicates a severe, internal error within the runtime system.
   - Masking underlying issues: Catching SystemException can obscure the true cause of the problem, making it difficult to diagnose and fix the underlying issue.

   instead:
   - Handle relevant exceptions: Identify and handle potential errors within your application using specific exception types like ArgumentException, NullReferenceException, or custom exceptions relevant to your code.
   - Provide Informative Error Messages: Offer clear and informative error messages to users or callers, explaining the issue and potential steps for resolution.
6. Yes, you generally want to avoid SystemException occurring in your application. While you can't directly control its occurrence as it's thrown by the runtime system, you can take proactive measures to minimize the likelihood of conditions that might lead to it:
   - Address Potential Errors Early
   - Practice Robust Coding Techniques
   - Thoroughly Test Your Code
   - Keep Runtime Environment Updated
   - Monitor for SystemExceptions

References

1. Gillis, A. S. (2022, June 6). *exception handling*. Software Quality. https://www.techtarget.com/searchsoftwarequality/definition/error-handling
2. Dotnet-Bot. (n.d.). *NullReferenceException Class (System)*. Microsoft Learn. https://learn.microsoft.com/en-us/dotnet/api/system.nullreferenceexception?view=net-8.0
3. More, S. (2023, April 15). *C# NullReferenceException*. EDUCBA. https://www.educba.com/c-sharp-nullreferenceexception/
4. Dotnet-Bot. (n.d.-a). *IndexOutOfRangeException Class (System)*. Microsoft Learn. https://learn.microsoft.com/en-us/dotnet/api/system.indexoutofrangeexception?view=net-8.0
5. Dotnet-Bot. (n.d.-c). *StackOverflowException Class (System)*. Microsoft Learn. https://learn.microsoft.com/en-us/dotnet/api/system.stackoverflowexception?view=net-8.0
6. *Newest "stack-overflow" questions*. (n.d.). Stack Overflow. https://stackoverflow.com/questions/tagged/stack-overflow?tab=Newest
7. Dotnet-Bot. (n.d.-c). *OutOfMemoryException Class (System)*. Microsoft Learn. https://learn.microsoft.com/en-us/dotnet/api/system.outofmemoryexception?view=net-8.0
8. Pedamkar, P. (2023, April 13). *C# OutOfMemoryException*. EDUCBA. https://www.educba.com/c-sharp-outofmemoryexception/
9. Dotnet-Bot. (n.d.-b). *InvalidCastException Class (System)*. Microsoft Learn. https://learn.microsoft.com/en-us/dotnet/api/system.invalidcastexception?view=net-8.0
10. *C# InvalidCastException - Dot Net Perls*. (n.d.). https://www.dotnetperls.com/invalidcastexception#:~:text=Avoiding%20casts.,older%20types%20for%20this%20purpose.
11. Dotnet-Bot. (n.d.-a). *DivideByZeroException Class (System)*. Microsoft Learn. https://learn.microsoft.com/en-us/dotnet/api/system.dividebyzeroexception?view=net-8.0

12. Dotnet-Bot. (n.d.-a). *ArgumentException Class (System)*. Microsoft Learn. https://learn.microsoft.com/en-us/dotnet/api/system.argumentexception?view=net-8.0

13. Dotnet-Bot. (n.d.-b). *ArgumentOutOfRangeException Class (System)*. Microsoft Learn. https://learn.microsoft.com/en-us/dotnet/api/system.argumentoutofrangeexception?view=net-8.0

14. Dotnet-Bot. (n.d.-i). *SystemException Class (System)*. Microsoft Learn. https://learn.microsoft.com/en-us/dotnet/api/system.systemexception?view=net-8.0