Name: Rakyan Satrya Adhikara

Student ID: 219548135

Practical task 4.1 Report

Task 1

a) *NullReferenceException*
- Possible situation that leads to the exception is when you try to access a member of a type whose value is **null.**
- Runtime system is in charge of throwing the exception.
  - You **should not** throw exceptions of this type since it usually mean that there are some actual mistakes in your logic.
- It can be generally caught by the *try-catch* block, with *NullReferenceException* as the main exception. It can be eliminated by for instance, declaring the number of elements in the array before initializing it.
- If the exception occurs, you should generally catch this exception type since this caused by some mistake in logic.
  - Reason: It may cause further damage to the program if the exception is being ignored
- this exception is one of the cases when you want to avoid this exception to occur in your application in general.
  - Your actions as a programmer to avoid it by catch the exception, analyze it and declare the variables before initializing it.

b) *IndexOutOfRangeException*
- Possible situation that leads to the exception is when an attempt is made to access an element of an array or collection with an index that is outside its bounds.
- Runtime system is in charge of throwing the exception.
  - You **should not** throw exceptions of this type since it usually means that there are some actual mistakes in your logic.
- It can be generally caught by the *try-catch* block, with *IndexOutOfRangeException* as the main exception. It can be eliminated by for instance, declaring the number of elements in the array before initializing it.
- If the exception occurs, you should generally catch this exception type.
  - Reason: It is a logic-related problem. Can be fixed by the user.
- This exception is a case when you want to avoid this exception to occur in your application in general.
  - Your actions as a programmer to avoid it by:
    1. Iterate the elements of the array using the *foreach* statement instead of iterating elements by index.
    2. Iterate the elements by index starting with the index returned by the *GetLowerBound* method and ending with the index returned by the *GetUpperBound* method.

3. If you are assigning elements in one array to another, ensure that the target array has at least as many elements as the source array by comparing their *Length* properties.

c) *StackOverflowException*

- Possible situation that leads to the exception is thrown when the execution stack overflows because it contains too many nested method calls. This class cannot be inherited.
- You as a programmer is in charge of throwing the exception.
  - You **should** throw exceptions of this type because it contains too many nested method calls. This class cannot be inherited.
- If you need to throw the exception, details would you provide as a message to the user are the class that is being constructed and number of classes that is being inherited.
  - Parameters would you include in the message is the class that caused the exception.
- The exception **cannot** be generally caught by try-catch exception.
- If the exception occurs, you should generally catch this exception type.
  - Reason: This might fail for class construction
- This exception is one of the cases when you want to avoid this exception to occur in your application in general.
  - Your actions as a programmer to avoid it by debug the code carefully and solve it by not using inheritance too often.

d) *OutOfMemoryException*

- 2 possible situations that leads to the exception:
  - You are attempting to expand a StringBuilder object beyond the length defined by its StringBuilder.MaxCapacity property.
  - The common language runtime cannot allocate enough contiguous memory to successfully perform an operation.
- Runtime system is in charge of throwing the exception.
  - You **should** throw exceptions of this type.
- If you need to throw the exception, details would you provide as a message to the user (caller) is the variable that caused the exception
  - Parameters would you include in the message is the variable that caused the exception.
- The exception can be generally caught by using *try-catch* block, with *OutOfMemoryException* as the main exception and to handle it by tried to allocate a large memory area at once.
- If the exception occurs, it is better to pass such exception to the user (caller).
  - Reason: An *OutOfMemoryException* doesn't mean that the memory is completely depleted, it just means that a memory allocation failed.
- This exception a case when you want to avoid this exception to occur in your application in general.

        - Your actions as a programmer to avoid it is by not to create objects that fill in that memory.

e) *InvalidCastException*

- Possible situation that leads to the exception is when the conversion of an instance of one type to another type is not supported.
- Runtime system is in charge of throwing the exception.
    - You **should** throw exceptions of this type since it means that you are doing something wrong or you are not checking for some invalid values first.
- If you need to throw the exception, details would you provide as a message to the user (caller) is the type of the conversion that is not supported.
    - Parameters would you include in the message is the variable that is going to be converted.
- The exception cannot be generally caught by using *try-catch* block, with *InvalidCastException* as the main exception. However, it should not be handled in a try/catch block. Instead, the cause of the exception should be eliminated.
- If the exception occurs, it is better to pass such exception to the user (caller)
    - Reason: it should not be handled in a try/catch block. Instead, the cause of the exception should be eliminated.
- This exception a case when you want to avoid this exception to occur in your application in general.
    - Your actions as a programmer to avoid it is by alternatively, you can use "as" keyword to cast and check if the result is null.

f) *DivideByZeroException*

- Possible situation that leads to the exception is when there is an attempt to divide an *integral* or *Decimal* value by zero.
- Runtime system is in charge of throwing the exception.
    - You **should** throw exceptions of this type means that you are doing something wrong or you are not checking for some invalid values first.
- If you need to throw the exception, details would you provide as a message to the user (caller) is the variable that cause the exception
    - Parameters would you include in the message is the variable that cause the exception
- The exception can be generally caught by using *try-catch* block, with *DivideByZeroException* as the main exception. However, it is better to avoid it.
- If the exception occurs, it is better to pass such exception to the user (caller)
    - Reason: To know which part causes this exception, so that it can be avoided later.
- This exception a case when you want to avoid this exception to occur in your application in general

        - Your actions as a programmer to avoid it is to make sure that the denominator in a division operation with *integer* or *Decimal* values is non-zero.

g) *ArgumentException*
    - Possible situation that leads to the exception is when a method is invoked and at least one of the passed arguments does not meet the parameter specification of the called method.
    - Runtime system is in charge of throwing the exception.
        - You **should** throw exceptions of this type but better not to cause this exception
    - If you need to throw the exception, details would you provide as a message to the user (caller) is the variable that got wrong argument
        - Parameters would you include in the message is the input argument that cause the exception
    - The exception can be generally caught by using *try-catch* block, with *ArgumentException* as the main exception. However, it is the best to avoid it.
    - If the exception occurs, you should generally catch this exception type
        - Reason: Not every user (caller) know what the argument is valid.
    - this exception a case when you want to avoid this exception to occur in your application in general
        - Your actions as a programmer to avoid it is to make sure that the argument provided is valid

h) *ArgumentOutOfRangeException*
    - Possible situation that leads to the exception is when a method is invoked and at least one of the arguments passed to the method is not *null* and contains an invalid value that is not a member of the set of values expected for the argument.
    - You as a programmer is in charge of throwing the exception
        - You **should not** throw exceptions of this type since it usually means that there are some actual mistakes in your logic.
    - The exception can be generally caught by using *try-catch* block, with *ArgumentOutOfRangeException* as the main exception. However, it is the best to avoid it.
    - If the exception occurs, it is better to pass such exception to the user (caller)
        - Reason: It is a developer error
    - This exception a case when you want to avoid this exception to occur in your application in general
        - Your actions as a programmer to avoid it is by eliminate the cause of the exception.

i) *SystemException*
    - Base class for system exceptions namespace.
    - You as a programmer is in charge of throwing the exception

        - You **should not** throw exceptions of this type since it simply does not contain any useful information.
- The exception can be generally caught by using *try-catch* block, with *SystemException* as the main exception. However, it is the best to make it more specific.
- If the exception occurs, you should generally catch this exception type.
    - Reason: It's better to catch and test which exception is it.
- This exception a case when you **do not** want to avoid this exception to occur in your application in general