Task05

Question: What is the purpose of the finally block?

The purpose of the finally block is to execute code that must run regardless of whether an exception occurs or not.

It is commonly used for cleanup operations such as closing files, releasing resources, or displaying a final message (like confirming that an operation has finished).

Even if an error happens in the try block or an exception is caught in the catch block, the finally block will always be executed.

Question: How does int.TryParse() improve program robustness compared to int.Parse()?

int.TryParse() improves program robustness compared to int.Parse() because it safely handles invalid input without throwing an exception. Instead of crashing the program when the input is not a valid integer, TryParse returns false and allows the program to handle the error gracefully, making it safer for user input or untrusted data.

Question: What exception occurs when trying to access Value on a null Nullable?

Accessing the .Value property of a Nullable<T> (or T?) variable that is **null** throws an:

InvalidOperationException

Example:

```
int? nullableNumber = null;
int number = nullableNumber.Value; // Throws InvalidOperationException
```

Message: "Nullable objects must have a value."

Question: Why is it necessary to check array bounds before accessing elements?

It is necessary to check array bounds before accessing elements to prevent IndexOutOfRangeException, ensure program stability, and avoid accessing invalid memory. Accessing an index outside the array's valid range (0 to length − 1) can crash the program or cause unpredictable behavior.

Question: How is the GetLength(dimension) method used in multi-dimensional arrays?

The GetLength(dimension) method returns the size of a specified dimension in a multi-dimensional array. The dimension is zero-based: 0 for the first dimension (rows), 1 for the second (columns), etc.

Question: How does the memory allocation differ between jagged arrays and rectangular arrays?

Rectangular arrays (`int[,]`) allocate a **single contiguous block** of memory for all elements, and all rows have the same length.

Jagged arrays (`int[][]`) allocate memory **row by row**, where each row is a separate 1D array, so rows can have different lengths.

Accessing jagged array elements is slightly slower due to pointer dereferencing, but jagged arrays are more flexible in row sizes.

Question: What is the purpose of nullable reference types in C#?

The purpose of nullable reference types in C# is to help developers detect and prevent null-related errors at compile time.

- By default, a reference type (like string) can be null, which may cause NullReferenceException at runtime.
- Declaring a reference type as nullable (string?) explicitly indicates that the variable may hold null.
- The compiler can then warn you if you try to use it without checking for null, improving code safety and reliability.

Question: Why must out parameters be initialized inside the method?
`out` parameters in C# must be initialized inside the method because they are intended to return values back to the caller. The compiler enforces this rule to ensure that the caller always receives a definite value, preventing the use of uninitialized variables.

Question: Why must optional parameters always appear at the end of a method's parameter list?

Optional parameters in C# must appear at the end of a method's parameter list because any arguments provided positionally are matched in order.

-If an optional parameter were in the middle, the compiler wouldn't know which arguments are meant for required parameters versus optional ones.
-Placing them at the end ensures that all required parameters are given first, and optional ones can be omitted safely.

Question: How does the null propagation operator prevent NullReferenceException?

The null propagation operator (?.) in C# prevents NullReferenceException by only accessing a member or element if the object is not null.

-If the object is null, the entire expression evaluates to null instead of throwing an exception.
-This allows safe chaining of member access without explicit null checks.

Question: When is a switch expression preferred over a traditional if statement?
A switch expression is preferred over a traditional `if` statement when you need to map a single value to multiple possible outcomes in a concise and readable way.

-Switch expressions are more concise than multiple `if-else if` statements.
-They return a value directly, making them useful in assignments.
-They improve readability for multiple discrete cases.

Question: What are the limitations of the params keyword in method definitions?

The limitations of the `params` keyword in C# are:

1. Only one `params` parameter per method
   -You cannot have multiple `params` arrays in a single method.

2. Must be the last parameter
   -The `params` parameter must come at the end of the parameter list.

3. Type restriction
   -The `params` parameter must be a single-dimensional array (e.g., `int[]`, `string[]`).