1 Part 01 - Theoretical Questions

1.1 Question 1: Purpose of the Finally Block

The `finally` block ensures that specific code executes regardless of whether an exception occurs or not. It is commonly used for cleanup operations such as closing files, releasing resources, or resetting states. Even if an exception is thrown and caught, or if the method returns early, the `finally` block will always execute.

1.2 Question 2: int.TryParse() vs int.Parse()

`int.TryParse()` improves program robustness by returning a boolean value indicating success or failure instead of throwing an exception. This allows for graceful error handling without the performance overhead of exception handling. `int.Parse()` throws a `FormatException` if the input is invalid, requiring try-catch blocks for error handling.

1.3 Question 3: Exception When Accessing Value on Null Nullable

When trying to access the `Value` property on a null `Nullable<T>`, an `InvalidOperationException` is thrown. This is why it's recommended to check `HasValue` before accessing `Value`, or use the null-coalescing operator (`??`) for safer access.

1.4 Question 4: Necessity of Checking Array Bounds

Checking array bounds before accessing elements prevents `IndexOutOfRangeException` at runtime. Array indices in C# are zero-based, so valid indices range from 0 to `array.Length - 1`. Accessing an invalid index can cause program crashes and unexpected behavior, making bound checking essential for program stability.

1.5 Question 5: GetLength(dimension) Method

The `GetLength(dimension)` method returns the number of elements in a specified dimension of a multi-dimensional array. For example, in a 2D array, `GetLength(0)` returns the number of rows, while `GetLength(1)` returns the number of columns. This is more flexible than the `Length` property, which returns the total number of elements.

1.6 Question 6: Memory Allocation Differences

Rectangular arrays allocate a contiguous block of memory for all elements, ensuring uniform access times. Jagged arrays are arrays of arrays, where each sub-array is allocated separately in memory. This allows for variable-length rows but results in non-contiguous memory allocation and potentially slower access due to additional pointer dereferencing.

1.7 Question 7: Purpose of Nullable Reference Types

Nullable reference types help prevent null reference exceptions by making the compiler aware of the nullability intent. They enable compile-time warnings when potentially null values are used without proper null checks, improving code safety. The feature distinguishes between references that can be null and those that cannot, leading to more robust code.

1.8 Question 8: Performance Impact of Boxing and Unboxing

Boxing converts a value type to a reference type by allocating memory on the heap and copying the value, which has performance overhead. Unboxing extracts the value type from the object, requiring type checking and copying. Frequent boxing/unboxing operations can cause performance degradation due to heap allocations and garbage collection pressure.

1.9 Question 9: Initialization of Out Parameters

Out parameters must be initialized inside the method because the compiler treats them as unassigned when entering the method. This ensures that the method always provides a value before returning, preventing the caller from receiving uninitialized data. The method is responsible for assigning a value to all out parameters before completion.

1.10 Question 10: Optional Parameters Position

Optional parameters must appear at the end of the parameter list because method calls match arguments to parameters positionally from left to right. If optional parameters were allowed before required ones, it would create ambiguity in determining which parameters are being passed, making the method call syntax unclear and error-prone.

1.11 Question 11: Null Propagation Operator Prevention

The null propagation operator (`?.`) prevents `NullReferenceException` by short-circuiting the evaluation when the left operand is null. Instead of attempting to access members on a null reference and throwing an exception, it returns null immediately. This provides a concise and safe way to chain member access on potentially null objects.

1.12 Question 12: Switch Expression vs If Statement

Switch expressions are preferred when mapping input values to output values in a concise, expression-based manner. They are more readable for simple value-to-value mappings, support pattern matching, and can be used in assignments or return statements directly. Traditional if statements are better for complex conditional logic with multiple statements per branch.

1.13 Question 13: Limitations of Params Keyword

The `params` keyword can only be used on the last parameter of a method, and only one `params` parameter is allowed per method. The parameter must be a single-dimensional array. While it provides flexibility in method calls, it can lead to ambiguity in method overload resolution and may have performance implications due to array allocation.