Part 1 – Questions & Answers

1. Question: What is the purpose of the finally block?
Answer: The finally block is used to make sure some code runs no matter what happens in the try block — whether there's an error or not. It's perfect for cleanup tasks, like closing files or showing a final message, because it always runs after try and catch.

---

2. Question: How does int.TryParse() improve program robustness compared to int.Parse()?
Answer: int.TryParse() is safer because it doesn't throw an exception if the input can't be converted to an integer. Instead, it just returns false, so you can handle invalid input smoothly without crashing the program.

---

3. Question: What exception occurs when trying to access Value on a null Nullable<T>?
Answer: If you try to access .Value when the nullable doesn't have a value, you'll get an InvalidOperationException.

---

4. Question: Why is it necessary to check array bounds before accessing elements?
Answer: Checking bounds avoids IndexOutOfRangeException. It makes sure you're not trying to read or write outside the valid positions in the array.

---

5. Question: How is the GetLength(dimension) method used in multi-dimensional arrays?
Answer: GetLength(dimension) tells you how many elements are in a specific dimension of the array. For example, GetLength(0) gives the number of rows, and GetLength(1) gives the number of columns.

---

6. Question: How does the memory allocation differ between jagged arrays and rectangular arrays?
Answer: A rectangular array is one solid block of memory, while a jagged array is an array of separate arrays. This means rows in a jagged array can be different lengths and stored in different places in memory.

---

7. Question: What is the purpose of nullable reference types in C#?
Answer: Nullable reference types help you write safer code by making it clear when a variable might be null. This way, the compiler can warn you about possible NullReferenceException issues before your program runs.

---

8. Question: What is the performance impact of boxing and unboxing in C#?
Answer: Boxing and unboxing slow things down because they involve extra work — boxing wraps a value type into an object, and unboxing unwraps it. This uses more CPU and memory, so avoiding it when possible is better for performance.

---

9. Question: Why must out parameters be initialized inside the method?
Answer: The whole point of out parameters is for the method to give back a value. That's why the compiler forces you to assign them a value inside the method before returning.

---

10. Question: Why must optional parameters always appear at the end of a method's parameter list?
Answer: Optional parameters go last so the compiler knows exactly which arguments you're providing. If they were in the middle, it could confuse which value belongs to which parameter.

---

11. Question: How does the null propagation operator prevent NullReferenceException?
Answer: The ?. operator stops the code from running further if the object before it is null. Instead of throwing an error, it just gives you null as the result.

---

12. Question: When is a switch expression preferred over a traditional if statement?
Answer: A switch expression is great when you're matching a single value against several known options. It's shorter, easier to read, and often cleaner than stacking multiple if/else if statements.

---

13. Question: What are the limitations of the params keyword in method definitions?
Answer: You can only have one params parameter in a method, and it must be the last one. Also, the parameter has to be an array type.