1. What is the purpose of the finally block?

The `finally` block contains code that always executes after a `try` (and optional `catch`), regardless of whether an exception was thrown or caught. It's commonly used for cleanup tasks like releasing resources.

---

2. How does `int.TryParse()` improve program robustness compared to `int.Parse()`?

`TryParse` returns a boolean indicating success or failure without throwing exceptions on invalid input, allowing safer parsing and better error handling, whereas `Parse` throws exceptions on invalid input.

---

3. What exception occurs when trying to access `Value` on a null `Nullable<T>`?

`InvalidOperationException` is thrown if you access `.Value` when the `Nullable<T>` has no value (`null`).

---

4. Why is it necessary to check array bounds before accessing elements?

To prevent `IndexOutOfRangeException`, accessing elements outside valid indices leads to runtime errors and possible program crashes.

---

5. How is the `GetLength(dimension)` method used in multi-dimensional arrays?

`GetLength(dimension)` returns the size (length) of the specified dimension (0-based) of a multi-dimensional array, allowing safe iteration over each dimension.

---

6. How does the memory allocation differ between jagged arrays and rectangular arrays?

- **Rectangular arrays:** Stored as a single contiguous block of memory.
- **Jagged arrays:** Arrays of arrays, each sub-array can have different lengths and are allocated separately on the heap.

7. What is the purpose of nullable reference types in C#?

Nullable reference types enable the compiler to detect and warn about potential `null` dereferences, improving code safety by differentiating nullable and non-nullable references.

---

8. What is the performance impact of boxing and unboxing in C#?

Boxing creates a heap object from a value type, and unboxing extracts it back. Both add overhead, including memory allocation and CPU time, thus can degrade performance if overused.

---

9. Why must out parameters be initialized inside the method?

Because the caller expects the `out` parameter to be assigned by the called method before returning, ensuring the variable is initialized when used afterwards.

---

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

To avoid ambiguity during method calls, since parameters after optional ones cannot be skipped without naming them explicitly.

---

11. How does the null propagation operator prevent `NullReferenceException`?

By short-circuiting evaluation if an object is `null`, it prevents access to members on `null` references, safely returning `null` instead.

Example:

```
string? name = person?.Name;
```

---

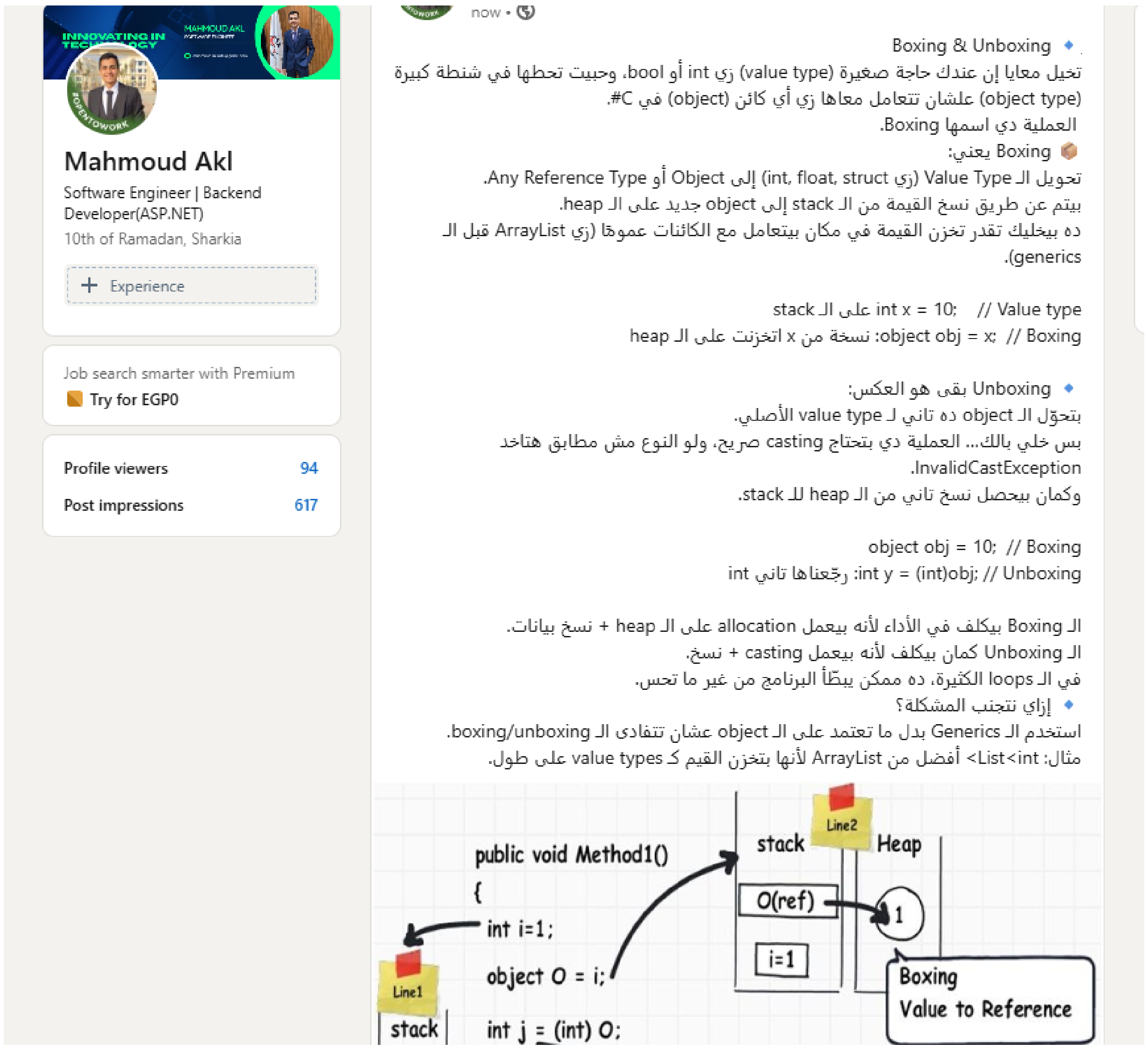
12. When is a switch expression preferred over a traditional if statement?

When you want a concise, readable way to select from multiple discrete cases or patterns and return a value directly.

---

13. What are the limitations of the `params` keyword in method definitions?

- Only one `params` array allowed per method.
- It must be the last parameter.
- All arguments passed to `params` must be of the declared element type or convertible to it.



Mahmoud Akl
Software Engineer | Backend Developer(ASP.NET)
10th of Ramadan, Sharkia

Job search smarter with Premium
Try for EGP0

Profile viewers	94
Post impressions	617

now •

🔹 Boxing & Unboxing
تخيل معايا إن عندك حاجة صغيرة (value type) زي int أو bool، وحبيت تحطها في شنطة كبيرة (object type) علشان تتعامل معاها زي أي كائن (object) في C#.
العملية دي اسمها Boxing.
📦 Boxing يعني:
تحويل الـ Value Type (زي int, float, struct) إلى Object أو Any Reference Type.
بيتم عن طريق نسخ القيمة من الـ stack إلى object جديد على الـ heap.
ده بيخليك تقدر تخزن القيمة في مكان بيتعامل مع الكائنات عمومًا (زي ArrayList قبل الـ generics).

int x = 10;   // Value type على الـ stack
object obj = x; // Boxing: نسخة من x اتخزنت على الـ heap

🔹 Unboxing بقى هو العكس:
بتحوّل الـ object ده تاني لـ value type الأصلي.
بس خلي بالك... العملية دي بتحتاج casting صريح، ولو النوع مش مطابق هتاخد InvalidCastException.
وكمان بيحصل نسخ تاني من الـ heap للـ stack.

object obj = 10; // Boxing
int y = (int)obj; // Unboxing: رجّعناها تاني int

الـ Boxing بيكلف في الأداء لأنه بيعمل allocation على الـ heap + نسخ بيانات.
الـ Unboxing كمان بيكلف لأنه بيعمل casting + نسخ.
في الـ loops الكتيرة، ده ممكن يبطّأ البرنامج من غير ما تحس.
🔹 إزاي تتجنب المشكلة؟
استخدم الـ Generics بدل ما تعتمد على الـ object عشان تتفادى الـ boxing/unboxing.
مثال: List<int> أفضل من ArrayList لأنها بتخزن القيم كـ value types على طول.

```
public void Method1()
{
  int i=1;
  object O = i;
  int j = (int) O;
```