Minutes of Session – C# Advanced Concepts

Date: 24th July 2025

Duration: 8Hrs

Topics Covered:

1. Exception Handling in C#
2. C# 7.0 & 8.0 Features
3. Searching & Sorting + Indexers
4. Quiz & Problem Solving

---

1. Exception Handling in C#

Key Concepts Covered

✔ `try-catch-finally`

- Handling runtime errors gracefully.
- `finally` block for cleanup (e.g., closing files).

✔ Common Exceptions

- `NullReferenceException`, `DivideByZeroException`, `ArgumentException`.

✔ Custom Exceptions

- Creating user-defined exceptions by inheriting from `Exception`.

```
public class InvalidAgeException : Exception

{

    public InvalidAgeException(string message) : base(message) { }

}
```

✔ Exception Filters

- Conditional catch blocks using `when`.

```
catch (Exception ex) when (ex.Message.Contains("invalid"))
{
    Console.WriteLine("Filtered exception caught!");
}
```

Instructor Demo

- Live coding: Handling file I/O exceptions.
- Best Practices:
  - Log exceptions with context.
  - Avoid empty catch blocks.

---

2. C# 7.0 & 8.0 Features

Key Features Explored

✔ Pattern Matching

- `is` keyword and `switch` expressions.

```
if (obj is string s) Console.WriteLine(s.Length);
```

✔ Tuples & Deconstruction

- Lightweight data structures.

```
var (name, age) = ("Alice", 30);
```

✔ Nullable Reference Types

- Compiler warnings for potential `null` references.

```
string? nullableString = null; // Explicitly nullable.
```

✔ Switch Expressions

```
var result = operation switch { "Add" => a + b, _ => 0 };
```

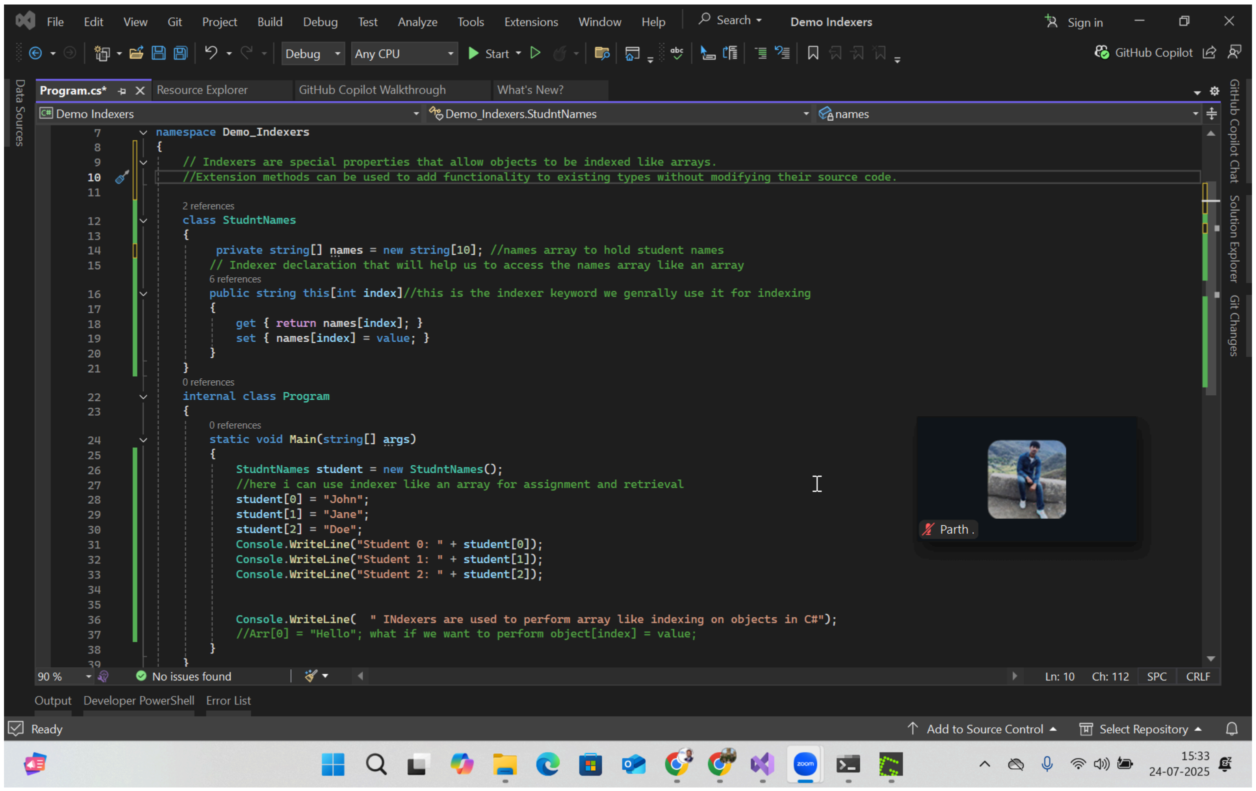
✔ Using Declarations

- Auto-disposal of resources.

```
using var file = new StreamReader("file.txt");
```

Guided Hands-On

- Exercise: Refactor legacy code using tuples and pattern matching.



```
namespace Demo_Indexers
{
    // Indexers are special properties that allow objects to be indexed like arrays.
    //Extension methods can be used to add functionality to existing types without modifying their source code.

    2 references
    class StudntNames
    {
         private string[] names = new string[10]; //names array to hold student names
        // Indexer declaration that will help us to access the names array like an array
        6 references
        public string this[int index]//this is the indexer keyword we genrally use it for indexing
        {
            get { return names[index]; }
            set { names[index] = value; }
        }
    }
    0 references
    internal class Program
    {
        0 references
        static void Main(string[] args)
        {
            StudntNames student = new StudntNames();
            //here i can use indexer like an array for assignment and retrieval
            student[0] = "John";
            student[1] = "Jane";
            student[2] = "Doe";
            Console.WriteLine("Student 0: " + student[0]);
            Console.WriteLine("Student 1: " + student[1]);
            Console.WriteLine("Student 2: " + student[2]);


            Console.WriteLine(  " INdexers are used to perform array like indexing on objects in C#");
            //Arr[0] = "Hello"; what if we want to perform object[index] = value;
        }
    }
```

---

3. Searching & Sorting + Indexers

Key Topics

✔ Sorting Algorithms

- Bubble Sort: Simple but inefficient ($O(n^2)$).
- Selection Sort: Finds min/max per iteration.
- Insertion Sort: Efficient for small datasets.

✔ Indexers vs Properties

- Indexers: Allow array-like access to objects.

```
public int this[int index] { get => data[index]; set => data[index] = value; }
```

- Properties: Encapsulate field access.

✔ Simple Attributes

- Metadata for classes/methods (e.g., `[Obsolete]`).
- Participants gained hands-on experience with advanced C# concepts.
- Common pitfalls (e.g., silent exceptions, `null` misuse) were addressed.