Minutes of Session – C# Advanced Concepts

Date: 25th July 2025

Duration: 8Hrs

Topics Covered:

Delegates in C#.NET	Delegates types in C# (Function Pointers)
	Single cast
	Multi cast
Standard C# Features	Anonymous method
	Lambda expression (What are different ways to use Lambda Expressions)
	Func
	Action
	Predicate
	Events
	Introduction to Reflection and its practical use(How it is implemented?)
	Generics
	Generic class
	Generic field
	Generic method
	Advantage of generics
	Threading
	Ref and Out keyword
	Async & Await(V.Imp)

C# 7.0	C#8.0
Extension methods	Nullable ref types , Ranges and Indices
Tuples and Deconstruction, Pattern matching , Out variables , Ref Local	Switch Expressions Read only members
Local functions	Using declarations, Async Stream
	Default interface methods
//Help an HR department build a lightweight	

employee information system using C# 7.0 features only
// Following are some user stories from a HR perspective:
//
//Store employee info(name, age, role)
//Filter employee based on role(using pattern matching)
//Extract and display just specific values(using tuples & Deconstruction)
//calculate experience using Local function
//Extend string data(Extension Methods)
// use out variables to validate age
//Ignore unwanted data using Discards
//use concise methods(Expression - Bodied members)	

Difference between Indexers and Properties ?

Milestone Assessment 120mins (single attempt)
Sat: 11: 00 AM - 01:00 PM
Pass: 70 %
Web Procoring enabled via web cam and tab proctoring | red flags

MCQ(30-35 MCQ) 30Mins	Platform based Coding Questions (2 or 3) 90 mins
Easy, Medium and Difficult Complexity	Easy, Medium and Difficult Complexity

Module 1: Delegates in C#.NET

Topics Covered:

- **Delegates**
    - *Definition*: A delegate is a type that represents references to methods with a particular parameter list and return type.

Syntax:

public delegate void MyDelegate(string msg);

MyDelegate del = new MyDelegate(MethodName);
del("Hello");

    -
    - *Types*: Single-cast, Multi-cast
    - *Real-Time Use*: Event handling (e.g., button click in WinForms)

- **Single-cast Delegate**
    - Holds reference to a single method.
- **Multi-cast Delegate**
    - Holds references to multiple methods. All are invoked in sequence.
    - Use += operator to add methods.

Best Practices:

- Always check for null (delegateName?.Invoke();)
- Use Func, Action, or Predicate if applicable for simplicity.

Interview FAQs:

- Difference between delegates and interfaces?
- When to use delegates instead of events?

Module 2: Lambda, Anonymous Methods, Func, Action, Predicate (1.5 Hours)

Anonymous Method

- Inline unnamed methods used with delegates.

Syntax:

delegate void Show(string message);

Show show = delegate(string msg) { Console.WriteLine(msg); };

Lambda Expression

- Shorter way of writing anonymous methods.

Syntax:

Action<string> display = msg => Console.WriteLine(msg);

Func, Action, Predicate

- Func<T, TResult>: Returns value
- Action<T>: No return
- Predicate<T>: Returns bool

Best Practices:

- Prefer lambdas over anonymous methods for readability.
- Use Func, Action, Predicate to simplify delegates.

Interview FAQs:

- Difference between Func and Action?
- Use case of Predicate?

Module 3: Events, Reflection, and Generics (1.5 Hours)

Events

- Special delegate that follows publish-subscribe pattern.

Syntax:
public event EventHandler MyEvent;

Reflection

- Runtime access to metadata and type information.

Syntax:
Type t = typeof(MyClass);
MethodInfo[] methods = t.GetMethods();

-
- **Real-time Use**: Dependency Injection, Unit Testing, ORM frameworks.

Generics

- Enables type-safe data structures.
- Generic Class, Method, Field.

  Syntax:

  public class MyGenericClass<T> { public T Value; }

Best Practices:

- Use generics to improve performance and reduce boxing/unboxing.
- Avoid excessive use of reflection in performance-critical code.

Interview FAQs:

- What is the purpose of reflection?
- Advantages of generics over object type collections?

Module 4: Threading, Ref & Out, Async & Await (1.5 Hours)

Threading

- Enables parallel execution.

Syntax:

```
Thread t = new Thread(MyMethod);
t.Start();
```

Ref and Out Keywords

- ref: Requires initialization before passing.
- out: Must be assigned inside the method.

Async & Await

- Non-blocking asynchronous programming.

Syntax:

```
public async Task<int> GetDataAsync() {
   await Task.Delay(1000);
   return 42;
}
```

-

Best Practices:

- Use ConfigureAwait(false) in library code.
- Always handle exceptions in async methods.

Interview FAQs:

- Difference between ref and out?
- When to use async and await?

Assignments (Case Studies)

Case Study 1: Student Report System (C# 8.0 Features)

User Story:
"As a school administrator, I want to create a report system to manage student records efficiently."

Topics Applied:

- Nullable Reference Types
- Async Streams
- Ranges and Indices
- Switch Expressions
- Default Interface Methods
- Pattern Matching Enhancements
- Using Declarations
- Readonly Members

Case Study 2: Smart Home Automation Console System

User Story:
"As a home automation engineer, I want to control appliances using console triggers and logs."

Topics Applied:

- Delegates (Single/Multi)
- Anonymous Method, Lambda Expression
- Action & Predicate
- Events
- Generics (Class, Method, Field)

- Reflection
- Async/Await
- Ref & Out Keywords

Case Study 3: Smart E-Commerce Analytics Engine

User Story:
"As a data analyst, I want to process and analyze order data for efficient business decisions."

Topics Applied:

- Exception Handling: try-catch-finally, Custom Exceptions, Filters
- C# 7.0 Features: Pattern Matching, Tuples, etc.
- Async data fetch (C# 8.0)
- Searching/Sorting (Linear, Binary, Bubble, Selection, Insertion)
- Indexers and Attributes
- Delegate-based Notifications