Minutes of Session

Date: 23rd July 2025

Session: Advanced C# Concepts – Delegates, Events, Collections & File I/O

Topics Covered

1. Delegates & Events

- **Definition:**
    - **Delegate:** A type-safe function pointer that holds reference to methods with a specific signature.
    - **Multicast Delegate:** A delegate that can hold references to more than one method.
    - **Event:** A wrapper over delegates, used to follow the publisher-subscriber model in C#.

Scenario	Benefit
Payment Callback	Asynchronous operation notification
Custom Sorting	Flexible logic injection
Plugin Framework	Loosely coupled architecture
UI Event Handling	Dynamic, event-driven actions

- **Best Practices:**
  - Use EventHandler or EventHandler<T> as the standard for event declarations.
  - Always check for null before invoking a delegate.
  - Use events for decoupling between components.
- **FAQs for Viva:**
  - Q1: What is a multicast delegate?
    A: A delegate that can reference multiple methods, executed in order.
  - Q2: How do events differ from delegates?
    A: Events are a restricted form of delegates, cannot be invoked outside the class that declares them.
  - Q3: What's the purpose of += and -= in event handling?
    A: To subscribe and unsubscribe event handlers.
  - Q4: What is the default delegate type for events?
    A: EventHandler or EventHandler<TEventArgs>
- **Demo:**
  - A Button Click Demo was created to simulate OnButtonClicked using events and a multicast delegate printing two messages.

---

2. Collections – Lists & Dictionaries

- **Definition:**
  - **List<T>:** A dynamic array capable of storing a collection of objects.
  - **Dictionary<TKey, TValue>:** A collection of key-value pairs for fast lookup.
- **Best Practices:**
  - Always check for null or existence before adding/removing from Dictionary.
  - Use foreach for iteration unless index-specific access is needed.

    - Choose Dictionary when fast lookups are essential.

- **FAQs for Viva:**
    - Q1: What is the difference between List<T> and Dictionary<TKey, TValue>?
      A: Lists are ordered collections, Dictionaries store data in key-value pairs with fast retrieval.
    - Q2: What exception is thrown if a key already exists in a Dictionary?
      A: ArgumentException.
    - Q3: How can we safely access an item in Dictionary?
      A: Use TryGetValue.

- **Demo:**
    - Created a basic Inventory System using List<Item> and Dictionary<string, Item> for stock management.

---

3. File I/O – Streams & Serialization

- **Definition:**
    - **Stream:** An abstraction for reading and writing bytes.
    - **Serialization:** Converting an object to a format (like JSON or XML) for persistence or transmission.

- **Best Practices:**
    - Always close or dispose streams using using block.
    - Use BinaryFormatter, XmlSerializer, or JsonSerializer as per requirement.
    - Avoid hard-coded file paths – use configuration.

- **FAQs for Viva:**
    - Q1: What are the types of Streams in C#?
      A: FileStream, MemoryStream, NetworkStream, etc.

- Q2: How do you serialize an object in C#?
  A: Using JsonSerializer.Serialize() or XmlSerializer.Serialize().

- Q3: What is the purpose of using using statement in File I/O?
  A: Ensures automatic resource disposal.

- **Lab:**
  - Data Export Lab – Students created a JSON file of product data exported from List.
  - Config File Parser – A sample .config file was parsed using ConfigurationManager.

Demo Scenario: File Handling

Scenario: Export Inventory to File and Load Back

- Create a class Item with properties: Name, Quantity, Price.
- Populate a List<Item> as in-memory data.
- Serialize this list to JSON using System.Text.Json and write to inventory.json.
- Clear the list, then deserialize the JSON file back into the list to verify persistence.

```
public class Item {

   public string Name { get; set; }

   public int Quantity { get; set; }

   public double Price { get; set; }

}


// Serialization

string json = JsonSerializer.Serialize(itemList);
```

```
File.WriteAllText("inventory.json", json);


// Deserialization

string readJson = File.ReadAllText("inventory.json");

List<Item> loadedList = JsonSerializer.Deserialize<List<Item>>(readJson);
```

- Demonstrated how to wrap the entire flow in a try-catch block, and used using for file stream management.

Session Summary

- Concepts covered with hands-on labs and real-world scenarios.
- Delegates and Events demonstrated with UI simulation.
- Inventory and File I/O gave practical insight into serialization and config management.
- Emphasis on coding best practices and robust exception handling.
