W1D3: Minutes of Session

Date: 21st July
Topic: Object-Oriented Programming (OOP) – Part 1
Duration: Full Day
Focus Areas:

- Classes & Objects
- Constructors
- Access Modifiers
- Class Diagram & Inheritance
- Shape Hierarchy
- Getters/Setters
- Static Members
- Calculator Class Demo

Concepts Covered

1. Classes and Objects

- **Definition:** A *class* is a blueprint for creating objects. An *object* is an instance of a class.

Syntax:

```
class Car {
   public string model;
}
```

```
Car myCar = new Car();

myCar.model = "BMW";
```

- **Best Practices:**
    - Use PascalCase for class names.
    - Keep class responsibilities focused (Single Responsibility Principle).SOLID principles
    - Avoid exposing fields; use properties instead.

---

2. Constructors

- **Definition:** A constructor is a special method that initializes objects.
- **Types:** Default constructor, parameterized constructor, static constructor.

Example:

```
public class Student {
   public string Name;
   public Student(string name) {
      Name = name;
   }
}
```

- **Best Practices:**

- Always initialize required fields in constructors.
- Use constructor chaining for reuse (: this(...)).

---

3. Access Modifiers

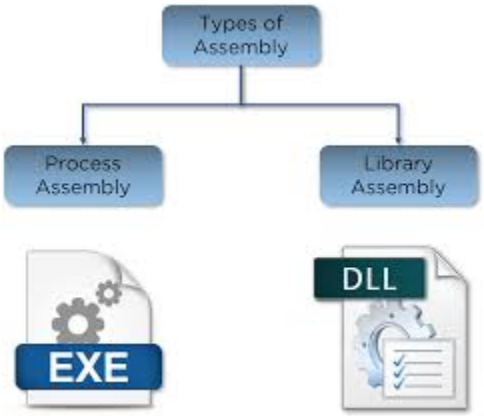
- **Definition:** These define the accessibility of classes and their members.
- **Types:**
  - public, private, protected, internal, protected internal, private protected

Example:

```
class Sample {
   private int x;
   protected void Show() { }
}
```

Caller's location	public	protected internal	protected	internal	private protected	private
Within the class	✔	✔	✔	✔	✔	✔
Derived class (same assembly)	✔	✔	✔	✔	✔	✘
Non-derived class (same assembly)	✔	✔	✘	✔	✘	✘
Derived class (different assembly)	✔	✔	✔	✘	✘	✘
Non-derived class (different assembly)	✔	✘	✘	✘	✘	✘

- **Best Practices:**
    - Restrict access to members as much as possible.
    - Use private by default, and expose only necessary methods.



---

4. Inheritance

- **Definition:** Inheritance allows a class (child) to acquire properties and behaviors of another class (parent).

Syntax:

```
class Animal {
   public void Speak() {
      Console.WriteLine("Animal sound");
```

```
    }
}

class Dog : Animal {
}
```

- **Best Practices:**
    - Favor composition over inheritance unless there is a true "is-a" relationship.
    - Avoid deep inheritance trees (prefer 1–2 levels).

---

5. Getters and Setters (Properties)

- **Definition:** Properties provide controlled access to fields.

```
public string Name { get; set; }
```

- **Best Practices:**
    - Use auto-properties for simplicity.
    - Add validation logic inside set if required.

---

6. Static Members

- **Definition:** Belong to the class rather than instances.

Syntax:

```
class MathHelper {
    public static double Pi = 3.14;
}
```

Best Practices:

Use static members for shared constants or utility methods.

- Avoid storing state in static members for instance-based logic.

Demos Conducted

Demo Title	Description
Shape Hierarchy	Demonstrated base class Shape and derived classes like Circle, Square showing inheritance and method overriding.
Class Diagram Creation	Created class diagrams using Visual Studio Class Designer.
Calculator Class	Designed a calculator with static methods and instance methods to perform operations like add, subtract, etc.
Getter/Setter Example	Used properties for encapsulating fields with validation.

Interview Questions Discussed

Beginner Level:

1. What is the difference between a class and an object?
2. What is a constructor in C#?
3. What is the default access modifier for class members? |

Intermediate Level:

1. Explain how inheritance works in C# with an example.
2. What is the difference between static and non-static members?
3. How are getters and setters implemented in C#?

Advanced Level:

1. When should you use private protected and protected internal modifiers?
2. Compare constructor overloading vs method overloading.
3. Why is it better to use properties over public fields?

Session Outcomes

- Students created their first full-featured C# class.
- Applied object-oriented principles in live code.
- Understood how to control access to class data using access modifiers.
- Practiced inheritance and class hierarchy design.
- Built a mini calculator project using static and instance members.

Next Steps

- Revise today's examples.
- Try modifying the Calculator class to add scientific functions.
- Prepare for **OOP Part 2** covering:
    - Method Overriding & Overloading
    - Polymorphism
    - Abstract Classes & Interfaces
    - Encapsulation in-depth