Minutes of Session

Date: 22nd July
Topic: Object-Oriented Programming – Part 2
Trainer: Parth Shukla
Duration: 08:45 AM to 5:30 PM
Participants: Wipro NGA- July 2025

Session Agenda:

1. **Polymorphism**
   - Function Overloading
   - Method Overriding
   - Abstract Classes vs Interfaces
2. **Contract Implementation**
   - Real-time Scenario: Payment Gateway Demo using Interfaces
3. **Exception Handling**
   - `try-catch-finally` block
   - Practical Scenarios and Best Practices
4. **File Handling in C#**
   - FileReader and StreamReader usage
   - Reading files with proper error handling
   - Industry-level use cases

Session Highlights & Demos:

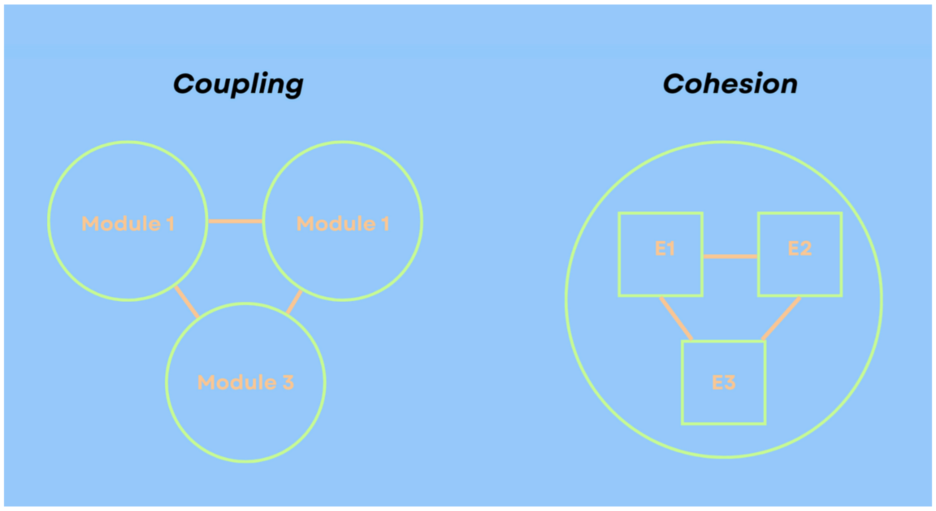
	`Abstract Class`	`Interface`
Inheritance	**Only one abstract class**	**Multiple interfaces**
Members	**We can have fields and constructors**	**Can not have fields or constructors**
Access Modifiers	**Support all access modifiers**	**Members are public default**
Use case	**Base class with shared code**	**Contract-based design**

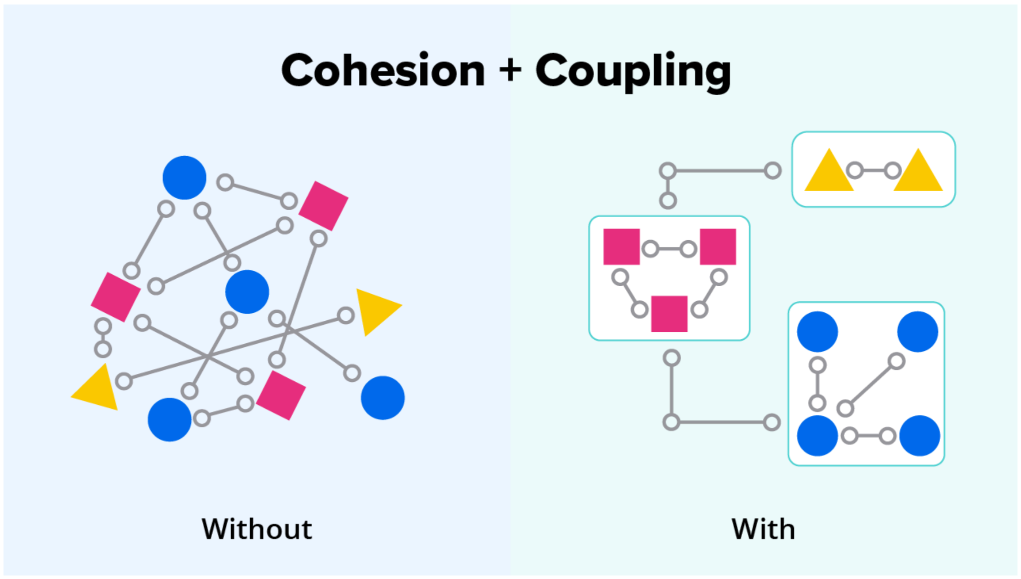
Animal Sound Demo – Understanding Polymorphism

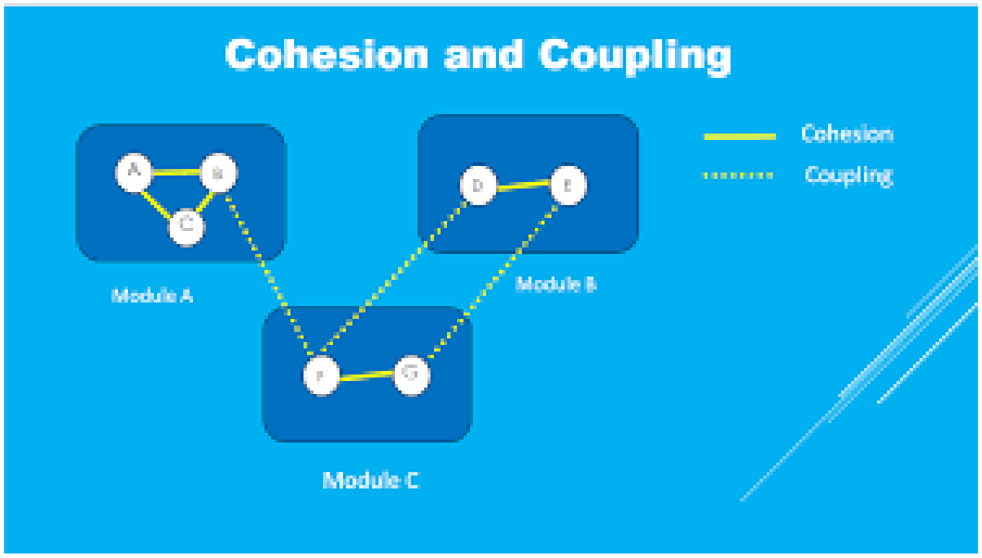
- **Overloading:** Demonstrated method overloading with different versions of `MakeSound()` in the `Animal` class.
- **Overriding:** Derived classes like `Dog` and `Cat` override the base method `MakeSound()`.
- **Outcome:** Learners understood how runtime and compile-time polymorphism works in practical code.

Interface & Abstract Class – Payment Gateway Demo

- Created `IPaymentGateway` interface with methods like `Pay()`, `Refund()`.
- Implemented multiple gateways like `CreditCard`, `UPI`, `PayPal`.
- Emphasized interface segregation and loose coupling.

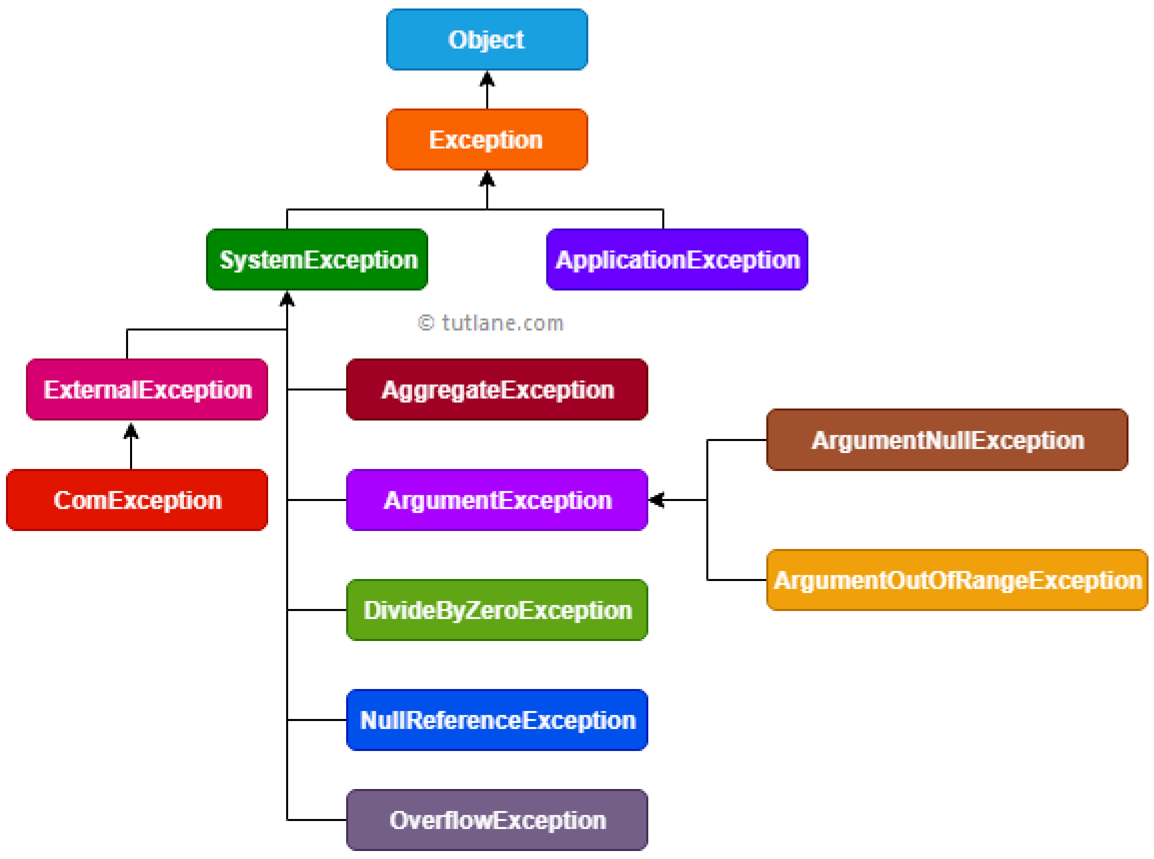

Coupling
Cohesion
Module 1
Module 1
Module 3
E1
E2
E3


Cohesion + Coupling
Without
With


Cohesion and Coupling
Cohesion
Coupling
A
B
C
D
E
F
G
Module A
Module B
Module C

Exception Handling

- Explained structured error management using `try-catch-finally`.
- Scenarios covered:
  - Division by zero
  - Null reference handling
  - Custom exception classes
- https://learn.microsoft.com/en-us/dotnet/api/system.exception?view=net-9.0



File Reader Demo

- Showcased usage of `StreamReader` and `File.ReadAllText()` for reading file contents.
- Ensured learners follow best practices:
  - Using `using` statement for auto-disposal

   - Checking file existence before access
   - Logging exceptions for diagnostics

Following are different ways to define path name for file handling in C#

Format	Example Code	Notes
Absolute	`@"C:\Path\To\File.txt"`	Fixed, specific location
Relative	`"folder\file.txt"`	Based on current project path
`Path.Combine`	`Path.Combine("folder", "file.txt")`	Best practice
Special Folder	`Environment.GetFolderPath(...)`	System folder (Desktop, Docs)
Current Directory	`Directory.GetCurrentDirectory()`	Dynamic during runtime
Network (UNC)	`@"\\Server\Share\file.txt"`	For shared network resources

Best Practices Covered

- Always override `ToString()` for better object representation.

- Favor interface-based coding for better testability and scalability.
- Never catch general `Exception` unless logging or rethrowing.
- Dispose unmanaged resources using `using` or `finally`.

Viva / Interview Questions Discussed

1. Difference between method overloading and overriding.
2. Can we override a static method in C#?
3. Abstract class vs Interface – when to use what?
4. What happens if `finally` throws an exception?
5. How to handle file not found scenario gracefully?
6. Can we have multiple catch blocks? Order of execution?
7. What if constructor throws an exception?

Assignments Given

- Implement your own `IFileLogger` and `IDatabaseLogger` interface demo.
- Create a polymorphic hierarchy for a `Vehicle` class with `StartEngine()` method overridden.
- Build a mini file reader app that handles user input and displays file contents.