In your OOP learning I find different type of classes like below: public Any other class public void Run() → Can be called anywhere private Only the same class private void Log() → Only usable inside its own class protected Same class + child classes protected void Init() → Inherited classes can use it internal Only within the same project internal void Debug() → Not accessible outside the project can explain each one of them to me with simple example and after that with a hard and useful example in a real company and combine different classes to see the usage of them

**Access Modifiers in C#: From Basics to Real-World Usage**

Let me explain each access modifier with simple examples first, then show you how they're used in a real company project.

**1. Simple Examples**

**(A)**public**- Fully Accessible**

csharp

Copy

Download

public class Car

{

public void StartEngine()

{

Console.WriteLine("Engine started!");

}

}

*// Usage:*

var myCar = new Car();

myCar.StartEngine(); *// Works everywhere*

**(B)**private**- Class-Only Access**

csharp

Copy

Download

public class Car

{

private void CheckOilLevel()

{

Console.WriteLine("Oil level OK");

}

public void StartEngine()

{

CheckOilLevel(); *// Only callable inside Car*

Console.WriteLine("Engine started");

}

}

*// Usage:*

var myCar = new Car();

myCar.StartEngine(); *// OK*

myCar.CheckOilLevel(); *// ❌ Compile error (private)*

**(C)**protected**- Inherited Classes Only**

csharp

Copy

Download

public class Vehicle

{

protected void LogDiagnostics()

{

Console.WriteLine("System check passed");

}

}

public class Car : Vehicle

{

public void StartEngine()

{

LogDiagnostics(); *// Works (inherited)*

Console.WriteLine("Engine started");

}

}

*// Usage:*

var myCar = new Car();

myCar.StartEngine(); *// OK*

myCar.LogDiagnostics(); *// ❌ Compile error (protected)*

**(D)**internal**- Project-Only Access**

csharp

Copy

Download

*// In Project A:*

internal class Logger

{

public void Log(string message) => Console.WriteLine(message);

}

*// In Project B (references Project A):*

var logger = new Logger(); *// ❌ Compile error (internal)*

**2. Real-World Company Example: E-Commerce System**

**Scenario**

We're building:

* A Product class (public API)
* A DiscountCalculator (internal business logic)
* A SecurityValidator (protected for extensions)
* A DatabaseConnection (private implementation detail)

**Code Implementation**

**(1) Public API (**ProductService**)**

csharp

Copy

Download

*// Public class (exposed to other projects)*

public class ProductService

{

private readonly DatabaseConnection \_db;

private readonly DiscountCalculator \_discounts;

public ProductService()

{

\_db = new DatabaseConnection();

\_discounts = new DiscountCalculator();

}

public Product GetProduct(int id)

{

var product = \_db.LoadProduct(id);

product.Price = \_discounts.ApplyDiscount(product.Price);

return product;

}

}

**(2) Internal Business Logic**

csharp

Copy

Download

*// Only usable within this project*

internal class DiscountCalculator

{

public decimal ApplyDiscount(decimal price)

{

return price \* 0.9m; *// 10% discount*

}

}

**(3) Protected Base Class (For Extensions)**

csharp

Copy

Download

public abstract class SecurityValidator

{

protected bool ValidateApiKey(string key)

{

return key == "COMPANY\_SECRET\_123";

}

}

*// Child class in another assembly*

public class CustomValidator : SecurityValidator

{

public bool CheckAccess(string key)

{

return ValidateApiKey(key); *// ✅ Works (protected)*

}

}

**(4) Private Implementation Detail**

csharp

Copy

Download

public class DatabaseConnection

{

private string \_connectionString = "Server=...";

private void LogQuery(string query)

{

Console.WriteLine($"Executing: {query}");

}

public Product LoadProduct(int id)

{

LogQuery($"SELECT \* FROM Products WHERE ID = {id}");

return new Product { Id = id, Name = "Laptop", Price = 999.99m };

}

}

**3. Why This Structure?**

| **Class** | **Access Modifier** | **Reason** |
| --- | --- | --- |
| ProductService | public | Exposed to web controllers |
| DiscountCalculator | internal | Business logic shouldn't leak outside |
| SecurityValidator | protected | Allow extensions but hide core logic |
| DatabaseConnection | private methods | Implementation details |

**4. Advanced Combined Usage**

csharp

Copy

Download

*// In a Web API Project*

[ApiController]

public class ProductsController : ControllerBase

{

private readonly ProductService \_productService;

public ProductsController()

{

\_productService = new ProductService();

}

[HttpGet("{id}")]

public IActionResult GetProduct(int id)

{

var product = \_productService.GetProduct(id);

return Ok(product);

}

}

**Access Flow**

1. **Public:** ProductsController calls ProductService (public)
2. **Internal:** ProductService uses DiscountCalculator (internal)
3. **Private:** DatabaseConnection hides its logging (private)
4. **Protected:** CustomValidator extends SecurityValidator (protected)

**5. Key Takeaways**

1. public → For APIs you want to expose widely.
2. private → Implementation details (change freely).
3. protected → Framework extensibility points.
4. internal → Team-only business logic.

**6. Real-Company Benefits**

* **Security:** Hide sensitive logic (DatabaseConnection).
* **Maintenance:** Change internals without breaking consumers.
* **Extensibility:** Allow sanctioned extensions (protected).
* **Teamwork:** Prevent misuse of business logic (internal).