Day 31/60 - .NET Series: Dependency Inversion Principle (DIP)

The Dependency Inversion Principle (DIP) states that high-level modules should not depend on low-level modules. Both should depend on abstractions. This promotes flexibility, scalability, and easier testing.

Why DIP Matters?
1. Loose Coupling: Reduces dependencies between classes.
2. Easier Testing: Mock dependencies for unit testing.
3. Scalability: Swap implementations without modifying core logic.

Example

Without DIP (Tightly Coupled Code)

```
public class EmailService
{
 public void SendEmail() { Console.WriteLine("Sending Email"); }
}

public class Notification
{
 private EmailService _emailService = new EmailService();
 public void Notify() { _emailService.SendEmail(); }
}
```

Here, Notification is tightly coupled to EmailService, making it hard to replace or test.

With DIP (Loosely Coupled Code)

```
public interface IMessageService { void SendMessage(); }

public class EmailService : IMessageService
{
 public void SendMessage() { Console.WriteLine("Sending Email"); }
}

public class Notification
{
 private readonly IMessageService _messageService;
 public Notification(IMessageService messageService) { _messageService = messageService; }
 public void Notify() { _messageService.SendMessage(); }
}
```

Now, Notification depends on an abstraction (IMessageService), allowing us to switch implementations easily.

DIP makes your .NET applications more maintainable and testable.

#dotnet #solid #dip #csharp