Understanding Dependency Injection in C#

What is Dependency Injection (DI)?

Dependency Injection is a design pattern that allows you to inject or provide the dependencies of a class from the outside rather than having the class create them itself.

This promotes loose coupling and enhances testability, flexibility, and maintainability of your code.

Basic Example: Injecting a Concrete Dependency

```
// Define the dependency interface
public interface IEngine
{
    void Start();
}

// Implement the dependency
public class PetrolEngine : IEngine
{
    public void Start()
    {
        Console.WriteLine("Petrol engine started.");
    }
}

// Dependent class that receives dependency via constructor
public class Car
{
    private readonly IEngine _engine;

    public Car(IEngine engine)
    {
        _engine = engine;
    }

    public void Drive()
    {
        _engine.Start();
        Console.WriteLine("Car is driving.");
    }
}

// Main program
class Program
{
    static void Main(string[] args)
    {
        IEngine engine = new PetrolEngine();
```

```
        Car car = new Car(engine);
        car.Drive();
    }
}
```

Line-by-Line Explanation

1. `IEngine` defines a common interface for engines.
2. `PetrolEngine` is a concrete implementation of the `IEngine` interface.
3. `Car` receives its dependency (engine) via constructor injection, making it loosely coupled.
4. The `Drive()` method uses the injected engine.
5. The `Main()` method creates a `PetrolEngine`, injects it into a `Car`, and then calls `Drive()`.

Advanced Example: Multiple Implementations with Swappable Engines

```
public class ElectricEngine : IEngine
{
    public void Start()
    {
        Console.WriteLine("Electric engine started.");
    }
}

// Main program with switchable engine types
class Program
{
    static void Main(string[] args)
    {
        IEngine engine;

        Console.WriteLine("Choose engine type: 1 - Petrol, 2 - Electric");
        var input = Console.ReadLine();

        if (input == "2")
            engine = new ElectricEngine();
        else
            engine = new PetrolEngine();

        Car car = new Car(engine);
        car.Drive();
    }
}
```

Line-by-Line Explanation

Understanding Dependency Injection in C#

1. `ElectricEngine` is another class that implements `IEngine`.
2. In `Main`, we prompt the user to choose an engine type.
3. Depending on the input, we instantiate the appropriate engine class.
4. We inject that engine into `Car` and call `Drive()`.
5. The car behaves the same regardless of which engine type is used, thanks to dependency injection.