# Assignment: Dependency Injection-Logger Class

Dependency injection describes a process whereby dependent objects are “injected” into a consuming object rather than having the consuming object instantiate them itself.

In this assignment, you will explore a simple example of dependency injection using logger classes. Please submit only **the completed program file (Program […].cs) to the drop box.**

## Overview

Dependency injection promotes flexibility and adaptability in software systems through **loose coupling**. Coupling refers to the degree to which objects are interdependent. Dependency injection eliminates strong dependencies between objects by using abstract interfaces to express dependencies.

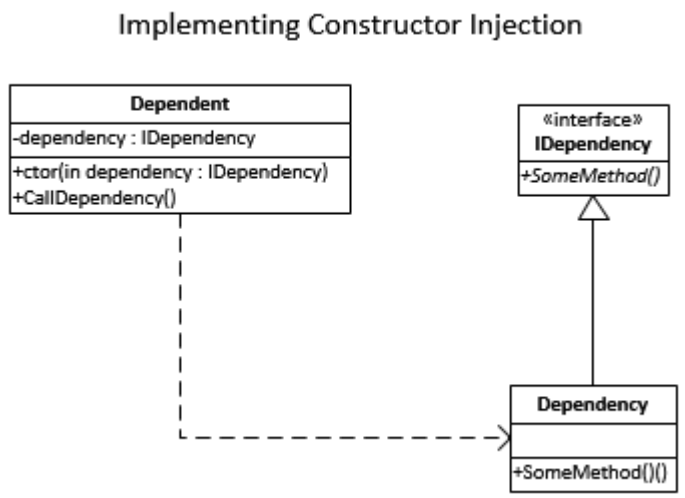
A dependency injection involves three elements:

- a consumer
- a contract
- an injector

The contract, an interface or abstract class, declares the requirements for the dependent object. The injector instantiates the dependent object and then injects it into the consumer. Typically, a variety of dependent objects satisfy the contract.

Constructor injection uses the consuming object’s constructor to inject the dependent object. Other alternatives include property setter and interface injection.

Here is a UML diagram describing constructor injection:




In this exercise, you will implement two simple logger classes, a console logger and a file logger, that implement a common logger interface. Other loggers could easily be defined (e.g., an event logger). Make that an exercise for next time.

# Instructions

## Steps to Follow

### Defining a Logger service

1. Define an interface named ILoggerService having two methods.

```
public interface ILoggerService
{
    void LogInfo(string msg, params object[] args);
    void LogError(string msg, params object[] args);
}
```

2. Define a class named ConsoleLogger implementing the interface ILoggerService.

```
public class ConsoleLogger : ILoggerService
{
    public void LogInfo(string msg, params object[] args)
    {
        Console.WriteLine("{0} INFO: {1}",
            DateTime.Now, string.Format(msg, args));
    }
    public void LogError(string msg, params object[] args)
    {
        Console.WriteLine("{0} ERROR: {1}",
            DateTime.Now, string.Format(msg, args));
    }
}
```

3. Define a class named FileLogger implementing the interface ILoggerService. Add a constructor that takes a filename parameter and saves it in a private field. Call the static AppendAllText method of the File class to log messages when implementing the LogInfo and LogError methods. Remember to import the System.IO namespace.

```
public class FileLogger : ILoggerService
{
    private string _fileName;
    public FileLogger(string fileName)...
    public void LogInfo(string msg, params object[] args)...
    public void LogError(string msg, params object[] args)...
}
```

## Defining an Injectable Bank Account class

1. Define a class named BankAccount having two read-only properties: ID (string) and Balance (decimal). Insert a parameterized, injectable constructor taking two parameters: id (string) and logger (ILoggerService).

```
public class BankAccount
{
    private ILoggerService _logger;

    public BankAccount(string id, ILoggerService logger)
    {
        _logger = logger;
        ID = id;

        logger.LogInfo("Creating Bank Account {0}", ID);
    }

    public string ID { get; private set; }
    public decimal Balance { get; private set; }
```

2. Add two more methods: void Deposit(decimal amount) and void Withdraw(decimal amount). Log incoming parameter values, before and after balances, and errors. Do not post a deposit or withdrawal if an error would result.

```
public void Deposit(decimal amount)
{
    _logger.LogInfo("Deposit requested: {0:c}", amount);

    if (amount <= 0)
    {
        _logger.LogError("Deposit rejected: must be positive.");
        return;
    }

    _logger.LogInfo("Old Balance: {0:c}", Balance);
    Balance += amount;
    _logger.LogInfo("New Balance: {0:c}", Balance);
}

public void Withdraw(decimal amount)...
```

### Testing the Logger Service

1. Write code to test the Console Logger and File Logger classes. Here's an example.

```
static void Main(string[] args)
{
    Console.Title = "Dependency Injection: Logger Example";
    Console.WindowHeight = (int)(Console.LargestWindowHeight * 0.75);
    Console.BufferHeight = 5000;

    // Test the Console Logger
    ILoggerService cLogger = new ConsoleLogger();
    BankAccount ba1 = new BankAccount("56782-04513", cLogger);
    ba1.Deposit(250m);
    ba1.Withdraw(100m);
    ba1.Withdraw(160m);

    // Test the File Logger
    ILoggerService fLogger = new FileLogger("56283-92465.txt");
    BankAccount ba2 = new BankAccount("56283-92465", fLogger);
    ba2.Deposit(1000m);
    ba2.Withdraw(300m);
    ba2.Withdraw(100m);
    ba2.Deposit(200m);

    Console.ReadLine();
}
```

## Summary

This assignment covered the concept of dependency injection. Two loggers, ConsoleLogger and FileLogger, were defined and injected into BankAccount objects. The logger method signatures were factored into a simple interface, ILoggerService. This approach allows more loggers to be defined and injected, e.g., IEventLogger, IDatabaseLogger, IXMLLogger, etc.

Dependency injection allows the separation of behavior from dependency making systems more flexible and extensible.