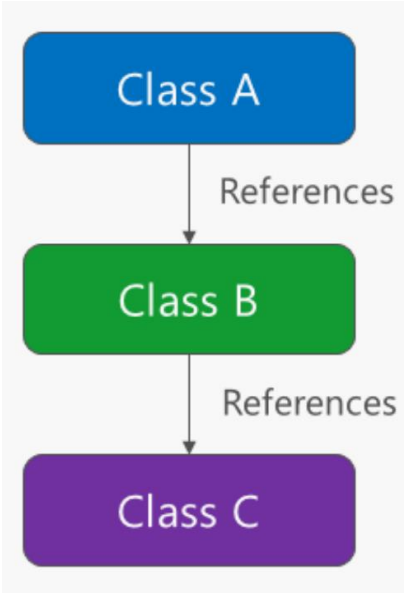
What is Dependency

- What is a *dependency?* It is an object that another object depends on
  - In an example below class `Test` creates and uses an object of class `SimpleInterest` i.e., class `Test` depends on class `SimpleInterest`



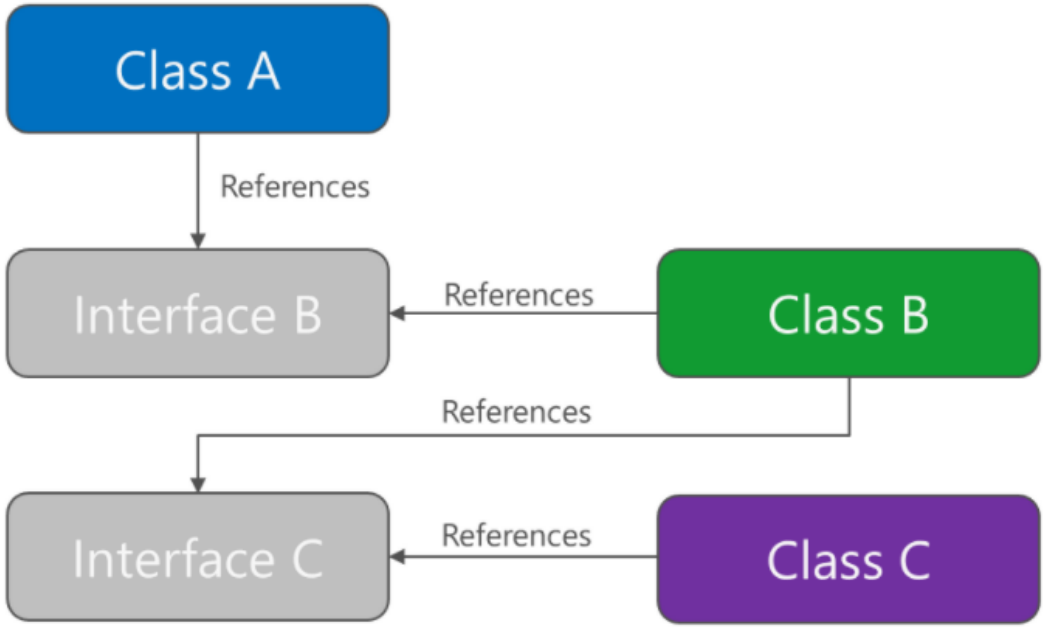

```
public class SimpleInterest
{
    public int Calculate(int p, int n, float r)
    {
        //...
        return 0;
```

```
class Test
{
    public void Start()
    {
        var obj = new SimpleInterest();
        var ans = obj.Calculate(1000, 2, 14.50f);
        Console.WriteLine(ans);
    }
}
```

Issues with Dependency

- Code dependencies are problematic and should be avoided for the following reasons.
  - To replace `SimpleInterest` with a different implementation like `CompoundInterest`, the `Test` class must be modified.
  - In a large project with multiple classes depending on `SimpleInterest`, the code becomes scattered across the app.
  - This implementation is also difficult to unit test.
  - See sample

Dependency Injection



- Dependency inversion is a key part of building loosely coupled applications.
- This is because, classes depend on interfaces rather than other classes.
- The resulting applications are more testable, modular, and maintainable as a result.

Dependency Injection (1/3)

- Dependency injection addresses these problems through:
  - The use of an interface or base class to abstract the dependency implementation.

```
public interface IBankInterest
{
    public int Calculate(int p, int n, float r);
}

public class SimpleInterest : IBankInterest
{
    public int Calculate(int p, int n, float r)
    {
        //...
    }
}
```

Dependency Injection (2/3)

- Registration of the dependency in a service container.
  - ASP.NET Core provides a built-in service container.
  - Services are typically registered in the Program.cs file.

```
// …
services.AddScoped<IBankInterest, SimpleInterest>();
```

Dependency Injection (3/3)

- Injection of the service into the constructor of the class where it's used.

```
public class Test
{
    private readonly IBankInterest _bankInterest;
    public Test(IBankInterest bankInterest)
    {
        _bankInterest = bankInterest;
    }

    public void Show()
    {
        _bankInterest.Calculate(10000, 2, 15.50f);
    }
}
```

The framework takes on the responsibility of creating an instance of the dependency and disposing of it when it's no longer needed.

DI & Its Advantages

- ASP.NET Core supports dependency injection (DI)
  - This is a technique for achieving Inversion of Control (IoC) between classes and their dependencies
- We do not use class `SimpleInterest` directly and instead use the interface `IBankInterest`.
- This makes it easy to change the implementation without modifying code throughout the project.
- We do not create instance of `SimpleInterest` ourselves, it is created by the DI container.