# Sebastian Canales Burke

K40 A01 Part B

# Part B Section 1 – Dependency Injection

On Moodle there is a sample MVC solution “HeritageCollege .NET 6 Target.zip”.

See [this tutorial](https://www.youtube.com/watch?v=9J9a77ga9R0) for samples and hints on how to implement dependency injection.

Load the solution “HeritageCollege”

Change the Student Controller to take in a IStudentRepository as a constructor.

Write out here the steps you had to do and explain each line (why are you doing this):

1. A screen shot of a computer program

   Description automatically generated
   1. Stated the reference to IStudentRepository so it is expecting the interface and not the service.
   2. Added the parameter in the constructor so it is the default, and sets the \_myRepo variable;
2. A screen shot of a computer program

   Description automatically generated  
   1. Made the interface public because it had an access modifier of internal, since the constructor is public, the parameters must also be.
3. A screen shot of a computer program

   Description automatically generated
   1. Registered the IStudentRepository and Student repository so that it is implemented when the controller is created.

Explain, what is the benefit of using Dependency Injection on the controller?

DI on the controller simplifies testing and promotes SRP and loose coupling.

Explain the different between the Singleton, Transient and the Scoped service lifetimes.

Singleton: Created once and is implemented for the entire length of the application lifetime.

Transient: New instance is created every time it is requested

Scoped: Created once per request or scope

You just created dependency injection on construction, what are all the types of dependency injection? *Hint: Read* [*this*](https://www.tutorialsteacher.com/ioc/dependency-injection)*.*

Constructor injection

Property injection

Method injection

Show an example of each highlighting the precise line that the dependency injection is happening.

CONSTRUCTOR

*public class CustomerService*

*{*

*CustomerBusinessLogic \_customerBL;*

*public CustomerService()*

*{*

*\_customerBL = new CustomerBusinessLogic(new CustomerDataAccess());*

*}*

*public string GetCustomerName(int id) {*

*return \_customerBL.ProcessCustomerData(id);*

*}*

*}*

PROPERTY

public class CustomerBusinessLogic

{

public CustomerBusinessLogic()

{

}

public string GetCustomerName(int id)

{

return DataAccess.GetCustomerName(id);

}

public ICustomerDataAccess DataAccess { get; set; }

}

public class CustomerService

{

CustomerBusinessLogic \_customerBL;

public CustomerService()

{

\_customerBL = new CustomerBusinessLogic();

\_customerBL.DataAccess = new CustomerDataAccess();

}

public string GetCustomerName(int id) {

return \_customerBL.GetCustomerName(id);

}

}

METHOD

interface IDataAccessDependency

{

void SetDependency(ICustomerDataAccess customerDataAccess);

}

public class CustomerBusinessLogic : IDataAccessDependency

{

ICustomerDataAccess \_dataAccess;

public CustomerBusinessLogic()

{

}

public string GetCustomerName(int id)

{

return \_dataAccess.GetCustomerName(id);

}

public void SetDependency(ICustomerDataAccess customerDataAccess)

{

\_dataAccess = customerDataAccess;

}

}

public class CustomerService

{

CustomerBusinessLogic \_customerBL;

public CustomerService()

{

\_customerBL = new CustomerBusinessLogic();

((IDataAccessDependency)\_customerBL).SetDependency(new CustomerDataAccess());

}

public string GetCustomerName(int id) {

return \_customerBL.GetCustomerName(id);

}

}

Further reading: [link](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-3.1)

Part B – Testing with Mock objects

This section continues with the HeritageCollege solution used in Part A.

Add->New Project->Unit Test Project, “HeritageCollegeTest”

In HeritageCollegeTest, select Dependencies, right click, Add Project Reference , under projects/solution select HeritageCollege.

Add a test for the StudentController

Rename UnitTest1 to StudentControllerTest

Add the HeritageCollege namespace

Add a test to just test the Index method:

[TestMethod]

public void TestIndexStudentRepo()

{

const int OUTPUT\_OVERHEAD = 2; // output has 2 extra newlines

// Arrange

StudentRepository testRepo = new StudentRepository();

StudentController testController = new StudentController(testRepo);

// Act

string testOutput = testController.Index();

// Assert

// count lines of output

Assert.AreEqual(2+ OUTPUT\_OVERHEAD, testOutput.Split('\n').Length);

// is Jean in there?

Assert.IsTrue(testOutput.Contains("Jean"));

Assert.IsTrue(testOutput.Contains("Richard"));

Assert.IsFalse(testOutput.Contains("Jose"));

}

1. Show me a snippet of the test running successfully (screen capture of the test run results)

*A screenshot of a computer

Description automatically generated*

Test driven development: Write a test that fails first, then add code until it works.

Add a test to test the StudentController.GetStudent() (hint: it doesn’t exist yet).

[TestMethod]

public void TestStudentGet()

{

// Arrange

StudentRepository testRepo = new StudentRepository();

StudentController testController = new StudentController(testRepo);

// Act

Student testStudent1 = testController.GetStudent(1);

Student testStudent3 = testController.GetStudent(3);

// Assert

Assert.IsNotNull(testStudent1);

Assert.IsNull(testStudent3);

}

Pass 1: throw the not implemented exception, test fails

1. Show me the output of the fail on the test run

*A screenshot of a computer

Description automatically generated*

Pass 2: Implement the code so that the test passes

1. Show me the output of the pass. Show me the code snippet of the updates in the Controller, IStudentRepository, and StudentRepository classes.

*A screenshot of a computer

Description automatically generated*

public Student GetStudent(int v)

{

return \_myRepo.GetStudent(v);

}

Build your own Mock.

Create your own new StudentRepository class, call it MyStudentRepository that implements the IStudentRepository, but under the hood, it only has 1 student in it (your name and studentId).

Create a new Unit test that uses MyStudentRepository, modelled like TestStudentGet() that validates the existence of studentId (yours) and one that doesn’t exist.

1. Show the code for MyStudentRepository.

*A computer screen shot of a program code

Description automatically generated*

1. Show the your test code

[Fact]

public void TestMyStudentRepostory\_GetStudent()

{

// Arrange

MyStudentRepository testRepo = new MyStudentRepository();

StudentController testController = new StudentController(testRepo);

// Act

Student testStudent1 = testController.GetStudent(1);

Student testStudent3 = testController.GetStudent(3);

// Assert

Assert.NotNull(testStudent1);

Assert.Null(testStudent3);

}

*A screenshot of a computer

Description automatically generated*

Congratulations, you manually created your own mock repository

*(let’s pretend the original StudentRepository had fancy Entity Framework internals).*

Now let’s see Moq in action.

Add the following unit test and ensure it runs. You may have to import/update the Mock package, for me, it was clicking through a wizard suggestion to fix.

// mocking the repository now

[TestMethod]

public void TestStudentGetWithMoq()

{

// Arrange

var testRepo = new Mock<IStudentRepository>();

testRepo.Setup(repo => repo.GetStudent(1)).Returns(new Student() { ID = 1 });

StudentController testController = new StudentController(testRepo.Object);

// Act

Student testStudent1 = testController.GetStudent(1);

Student testStudent3 = testController.GetStudent(3);

// Assert

Assert.IsNotNull(testStudent1);

Assert.IsNull(testStudent3);

}

1. Show me the output of the test run. Everything should pass

A screenshot of a computer

Description automatically generated

Your turn, create another test method that has a mocked repository with one entry (yours).

Validate you can get your studentId and the first name matches what you expect it to be.

1. Show me the output of the pass. Show me the code snippet of the updates in the new test method

A screenshot of a computer

Description automatically generated

// mocking the repository now

[Fact]

public void TestMyStudentRepository\_GetWithMoq()

{

// Arrange

var testRepo = new Mock<IStudentRepository>();

testRepo.Setup(repo => repo.GetStudent(1)).Returns(new Student() { ID = 1, FirstName = "Sebastian", LastName = "Canales Burke" });

StudentController testController = new StudentController(testRepo.Object);

// Act

Student result = testController.GetStudent(1);

// Assert

Assert.NotNull(result);

Assert.Equal("Sebastian", result.FirstName);

}

Further reading: [link](https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/testing?view=aspnetcore-3.1)

**To submit for Part B:**

Upload this completed document, ***UName\_*K40\_A1B.docx,** to Moodle with the answers filled in.

* Ensure all of your student answers are in blue.
* Ensure you are submitting the right file.
* Download the file you uploaded to Moodle, check it for accuracy and completion. You are responsible for the accuracy and completeness of this file.
* Ensure you submit well in advance of the deadline to avoid any user or technical errors.