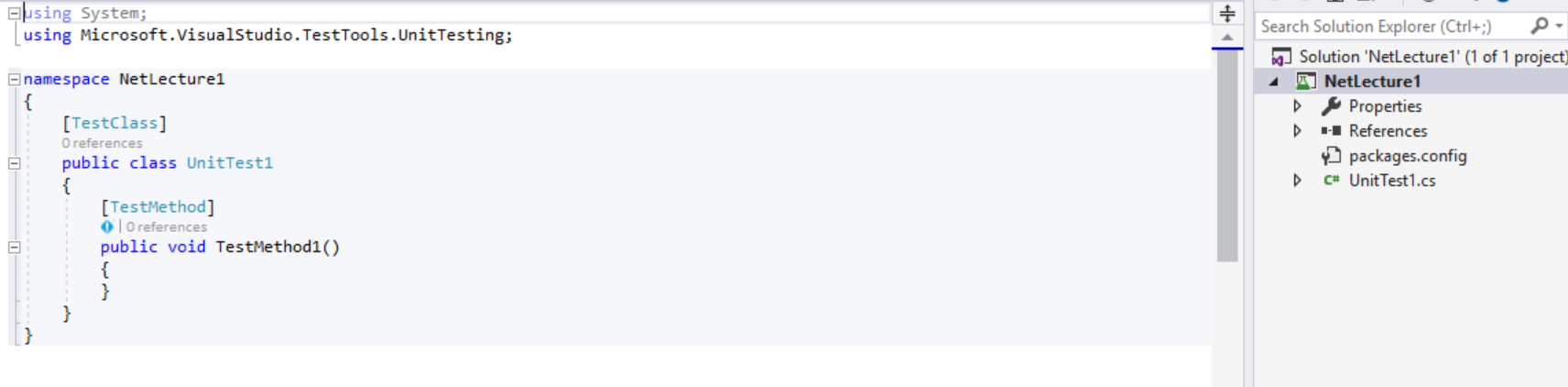
**Lecture done with Khurram**

**In this example, I am simply opening a browser with ‘bbc.c.uk’ by registering web driver to ‘Specflow ‘container and will be using context injection.**

1. Open VS
2. Create a new unit test (.net framework) project
3. Select project location and name and leave the checkbox unchecked that says ‘place solution and project files in the same location’
4. When the project is created, it looks as below:



1. Delete the UnitTest1.cs class from project explorer
2. Make sure the Specflow extension has been added for the visual studio by going into Extensions > Manage Extensions
3. Install the required browsers on the machine where automation will be run.
4. Install the following add-ons

Chrome Add-ons:

Xpath Helper/Xpath Checker- To validate the Xpath generated manually is correct & it identifies the correct Web Element

FireBug Lite-To Inspect Element

1. Next, install the following NuGet packages
   1. Nunit
   2. Nunit adapter
   3. Selenium Webdriver
   4. Selenium.Support
   5. Selenium.Chrome.Webdriver
   6. Specflow.Nunit
   7. Specflow.tools.Msbuild.Generation
   8. Specflow.Assist.Dynamics
   9. Nunit Console
2. Add the app.config file if missing by going into New Item > Application Configuration file
3. Add the following folders in the project
   1. Core
   2. Features
   3. StepsDefs
   4. PageObjects
   5. Resources
4. Create a new feature file e.g. BBCHome as below:

Feature: BBCHome

Scenario: Launch the BBC Home page

Given the user has launched the BBCHome page

1. Right click the step and select ‘Generate step definition’ and then select ‘copy methods to clipboard’
2. Add a new step definition class to map feature to the steps class by right clicking the ‘StepDefs’ folder and select new item and then add a steps class as below:

public class BBHomeSteps:BaseSteps

{

[Given(@"the user has launched the BBCHome page")]

public void GivenTheUserHasLaunchedTheBBCHomePage()

{

Driver.Navigate().GoToUrl("https://www.bbc.co.uk");

}

}

1. ‘Driver’ in the above code is not recognized yet.
2. In order to recognize the ‘Driver’ object above, we need to create another class called ‘BaseSteps’

public class BaseSteps

{

protected IWebDriver Driver;

public BaseSteps(IWebDriver driver)

{

Driver = driver;

}

}

1. After adding the ‘BaseSteps’ class above, BBCHomeSteps class would become like this:

public class BBHomeSteps:BaseSteps

{

public BBHomeSteps(IWebDriver driver) : base(driver)

{ }

[Given(@"the user has launched the BBCHome page")]

public void GivenTheUserHasLaunchedTheBBCHomePage()

{

Driver.Navigate().GoToUrl("https://www.google.co.uk");

}

}

* 1. In the class above, when ‘BBCHomeSteps’ object is created, constructor runs where we are passing ‘IWebDriver’ as argument. So this can be passed to the constructor of parent class (BaseSteps)
  2. The reason I am setting ‘Driver’ object in a parent class (BaseSteps) so it can be easily accessible in all child steps class (we may create). Setting ‘Driver’ in one parent class is easier than managing/maintaining in various steps classes.
  3. At this point ‘Driver’ object is now recognized.
  4. Check the project is building.

1. Until now, we have created a feature file and mapped it to a step class. We have also created a driver object in parent step class that is now accessible in the child steps class. But have we initialled the driver object yet. The answer is ‘No’
2. ‘IWebDriver’ is an interface and we have to tell which driver object we want at run time. This is where we need to create another class called ‘DriverFactory.cs or SetupWebDriver.cs’.
3. The ‘SetupWebDriver’ class looks as below:

[Binding]

class SetupWebDriver

{

private IWebDriver \_driver;

private readonly IObjectContainer \_container;

public SetupWebDriver(IObjectContainer container)

{

\_container = container;

}

[BeforeScenario]

public void initialize()

{

\_driver = getDriver("CHROME");

}

public IWebDriver getDriver(String requiredDriver)

{

switch (requiredDriver)

{

case "CHROME":

ChromeOptions options = new ChromeOptions();

\_driver = new ChromeDriver(options);

\_container.RegisterInstanceAs(\_driver);

break;

}

return \_driver;

}

}

* 1. Make sure the ‘SetupWebDriver’ class has a [Binding] attribute on the top
  2. By default, ‘Specflow’ uses a container called ‘IObjectContainer’ where all the instances are stored when we run the project.
  3. Now, what if I want to create a new object of IWebDriver? I will have to create the object, register it with the container and then pass as dependency to the binding steps classes as a constructor argument. This is called Dependency Injection (DI)

1. Finally, build the project and run the test in test explorer.

**Lecture 3 starts here (Lesson 4 is alternate to this lesson where I will save table object in \_context but in a different way)**

1. Next, I will implement the concept of context injection
2. Edit the feature file so the new file looks like this

Feature: BBCHome

Scenario: Launch the BBC Home page

Given the user has launched the BBCHome page

When the user enters the BBC new employee details

| *Name* | *Age* | *Phone* | *Email* |

| Imran | 41 | 999999 | test@test.com |

Then emplyee details can be seen at the employee view page

1. Implement the new steps in the step definition file and it looks like this. I have saved the full specflow data table in ScenarioContext and retrieved afterwards.

namespace NetLecture1.StepDefs

{

[Binding]

public class BBHomeSteps : BaseSteps

{

private readonly ScenarioContext \_context;

//driver is already registered with container in SetupWebDriver class

//Here we are only injecting it through constructor

public BBHomeSteps(IWebDriver driver, ScenarioContext context) : base(driver)

{

\_context = context;

}

[Given(@"the user has launched the BBCHome page")]

public void GivenTheUserHasLaunchedTheBBCHomePage()

{

Driver.Navigate().GoToUrl("https://www.bbc.co.uk");

}

[When(@"the user enters the BBC new employee details")]

public void WhenTheUserEntersTheBBCNewEmployeeDetails(Table table)

{

var name = table.Rows[0]["Name"];

var age = table.Rows[0]["Age"];

var phone = table.Rows[0]["Phone"];

var email = table.Rows[0]["Email"];

//the line below will add a new employee - demo and not functional

//employeePageObject.addNewEmployee(name, age, phone,email);

\_context["EmpDetails"] = table;

}

[Then(@"emplyee details can be seen at the employee view page")]

public void ThenEmplyeeDetailsCanBeSeenAtTheEmployeeViewPage()

{

Table table = \_context.Get<Table>("EmpDetails");

var name = table.Rows[0]["Name"];

var age = table.Rows[0]["Age"];

var phone = table.Rows[0]["Phone"];

var email = table.Rows[0]["Email"];

//employeePageObject.validateEmployee(name, age, phone,email);

}

}

}

**Note:**

* The benefit of context injection is to share data across steps within same scenario
* Scenario context is already registered with container
* Resolve method returns the ScenarioContext instance which is already registered as default with specflow container
* IEnumerable is used for simple iteration over the collection (not used in this example)
* Make sure to add Newtonsoft dependency for JSonConvert
* I have not implemented any page objects in this example – just using in the code for demo only, no code has been written for page objects