public class  BusinessLogic  
{ /\*Here we can write a getGile() method to iterate files in the mentioned directory path      i.e /feed-products to get extension of files and create respective class Object in If Else condition block.\*/

IFileType interfaceObj;

FileFactory factoryObj = new FileFactory();

public BusinessLogic(string InputFileType)

{

this.interfaceObj = factoryObj.getFile(InputFileType)

}

public static void main(string[] args)

{

BusinessLogic mainClassObj1= new BusinessLogic(“yaml”),

mainClassObj1.importFile();

BusinessLogic mainClassObj2= new BusinessLogic(“json”),

mainClassObj2.importFile();

}

}

}

Public Class FileFactory{

public static IFileType getFile(string type){

if (type==json){

return new JsonFile();

}

else If(type==yaml){

return new YamlFile();

}

}

}

public interface IFileType

{

void importFile();

}

public class JsonFile: IFileType

{ IDictionary data = new Hashtable();

public void importFile()

{

//way to read and import yaml File and

data.add(“tags”, value);

data.add(“name”, value);

data.add(“twitter, value”)

//pass value to DBFunction

DBFunction DBObj= new DBFunction();

DBObj.ImportFileinDB(data);

}

}

public class yamlFile: IFileType

{ IDictionary data = new Hashtable();

public void importFile()

{

//way to read and import yaml File and

data.add(“categories”, value);

data.add(“title”, value);

data.add(“twitter, value”)

//pass value to DBFunction

DBFunction DBObj= new DBFunction();

DBObj.ImportFileinDB(data);

}

}

Public Class DatabaseConnection

{

Private static DBConnection(){

try

{

//get Database connection

return connection;

}

catch(Exception e){

log.Error(er);

return null;

}

}

public DataTable ExecuteQuery(SQLCommand cmd, String action)

{

try

{

//use connection

Switch(action)

{

// switch cases for INSERT, UPDATE, SELECT etc queries

case “SELECT”:

//open connetion,

//execute Query

 return dataTable;

case “INSERT”:

//open connetion

 //execute Query

  break;

}

}

finally

{

//close connection

}

}

public class DatabaseFunction:DatabaseConnection

{

/\*contains methods where we are writing query/Commands and passing them as parameters to call method ExecuteQuery() in Parent Class\*/

// return ExecuteQuery(cmd, “INSERT”);

}

Unit Test Cases

GetFileUnitTest.cs

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System;

namespace GetFileUnitTest

{

    [TestClass]

    public class GetFileUnitTest

    {

        [TestMethod]

        public void TestMethod\_WhenCsv\_ReturnedInvalid()

        {   //Arrange

            string input4 = "csv";

            //Act

            string output4 = GetFileFromDirectory.GetFileClas.GetFile(input4);

            //Assert

            Assert.AreEqual("Invalid Input", output4);

        }

        [TestMethod]

        public void TestMethod\_WhenJson\_ReturnedJson()

        {

            //Arrange

            string input2 = "json";

            //Act

            string output2 = GetFileFromDirectory.GetFileClas.GetFile(input2);

            //Assert

            Assert.AreEqual("json", output2);

        }

        [TestMethod]

        public void TestMethod\_WhenEmpty\_ReturnedInvalid()

        {

            //Arrange

            string input3 = " ";

            //Act

            string output3 = GetFileFromDirectory.GetFileClas.GetFile(input3);

            //Assert

            Assert.AreEqual("invalid input", output3);

        }

        [TestMethod]

        public void TestMethod\_WhenYaml\_ReturnedYaml()

        {

            //Arrange

            string input = "yaml";

            //Act

            string output = GetFileFromDirectory.GetFileClas.GetFile(input);

            //Assert

            Assert.AreEqual("yaml", output);

        }

    }

}

GetFileClass.cs

using System;

namespace GetFileFromDirectory

{

    public class GetFileClass

    {

        public static string GetFile(string input)

        {

            if (input == "yaml" || input == "json")

            {

                return input;

            }

            else return "invalid input";

        }

    }

}

Explanation

Classes are designed using IOC pattern while keeping in mind SOLID principles of OOPS as per my knowledge. Code Execution starts with BusinessLogic Class. Design has one common Interface IFileType and concrete classes JsonFile, YamlFile implementing the interface so that classes have their own way to read and import data. There is a class FileFactory which we use to get the type of object we need. In BusinessLogic we use FileFactory to get a IFileType Object.

Unit testing is done on MSTest platform for the first time. All Test cases are being passed and is written to check scenario to perform import operation only for required file extensions eg. json, yaml here. GetFileUnitTest.cs file runs test cases and has a reference for GetFileClass.cs.

If I had more time I would have implemented IOC for which DataAccessLayer i.e. for SQL or MongoDB because in future switching back would have been easy and I would have tried performing Functional Testing if were given more time.