# Exercise: Intro to Classes and Interfaces

The goal of this assignment is to help you learn how to define, implement, and use interfaces. You will be creating 4 classes (MapDocument, Map, Layer and FeatureLayer) and 5 interfaces (IMapDocument, IMapManager, IMap, ILayer and IFeatureLayer). The classes and interfaces are outlined in the following table.

In the following table, the syntax follows the conventions you will find in the top left corner of ArcObjects Object Model Diagrams (e.g. <http://resources.arcgis.com/en/help/arcobjects-net/pdf/CartoObjectModel.pdf>)

For properties, the syntax is

PropertyName : Type.

The syntax for methods is

MethodName(paramName : ParamType) : ReturnType

If the method does not return anything (i.e. void), it will not be included, for example

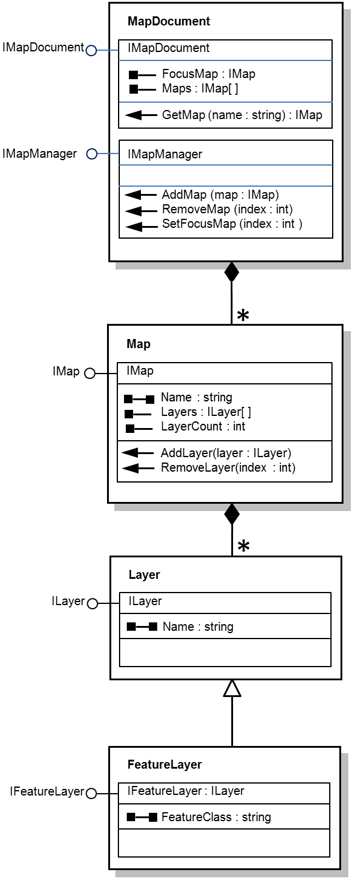
VoidMethodName(paramName : ParamType)

|  |  |  |
| --- | --- | --- |
| **MapDocument** Class | | Notes |
| IMapDocument | ***Properties:*** |  |
| FocusMap : IMap | Read only |
| Maps : IMap[] | Read only |
| ***Methods:***  GetMap(name : string) : IMap |  |
| IMapManager | ***Methods****:* |  |
| AddMap(map : IMap) |  |
| RemoveMap(index : int) |  |
| SetFocusMap(index : int) | Set the map that will have focus (i.e. active map) |
| **Map** Class | |  |
| IMap | ***Properties:*** |  |
| Name : string | Read/Write |
| Layers : ILayer[] | Read only |
| LayerCount : int | Read only |
|  |  |
| ***Methods:*** |  |
| AddLayer(layer : ILayer) |  |
| RemoveLayer(index : int) |  |
|  |  |  |
| **Layer** Class | |  |
| ILayer | ***Property:*** |  |
| Name : string | Read/Write |
| **FeatureLayer** Class | |  |
| IFeatureLayer | ***Property:*** |  |
| FeatureClass : string | Read/Write |

## PART I: Understand the OMD

On the following page is a sketch of the class diagram of the above classes and the relationships between them. Use the ESRI key at the end of this document to help understand what the class diagram is conveying. Note the following relationships between the classes:

* A MapDocument is composed of one or more Maps
* A Map is composed of zero or more Layers.
* A FeatureLayer is derived from Layer (FeatureLayer is a type of Layer = Type Inheritance)
* Layer is an Abstract class and IFeatureLayer inherits ILayer



## PART II: Code classes, interfaces, and methods

1. Download and follow the instructions in <https://github.com/viljoed/gis4x07/raw/master/GIS4x07_ExerciseSetup.docx> to create a Git repository for this exercise. Make sure the name is gis4207-day10.
2. Download and follow the instructions in <https://github.com/viljoed/gis4x07/raw/master/VS_SolutionSetup.docx> to create the Visual Studio solution for this exercise.
3. The solution name is MyProGis.sln
4. Open the MyProGis.sln solution
5. Add a Class Library Project called MyProGisBLL
6. Add a Test project called MyProGisBLLTests.
7. Add all interfaces to the MyProGisBLL class library project (one file per interface) Please note, when you add and item to the project, Interface is one of the options. Don’t add any interface members (properties or methods) yet. Build your code and correct any errors.
8. Make the scope of all the interfaces public. Build your code and correct any errors.
9. Add all members (properties and methods) to the interfaces. The interface members are implicitly public so do not include a scope keyword for the members. Remember, interfaces have no implementation code. For example, a function that returns an integer will have the following syntax in an interface:  
     
    int GetSomeNumber(string someParam);  
     
   Note there are no { } for method members in an interface.  
     
   A read-write property will have the following syntax   
     
    int SomeProperty {get;set;}
10. Build your code and correct any errors.
11. Add all classes to the MyProGisBLL class library project. Build your code and correct any errors.
12. Make the scope of all the classes public.
13. Add the base class (if any) and interface names to the class statements (statements that contain the class keyword). For example, a FeatureLayer is a type of layer and the FeatureLayer class implements IFeatureLayer  
      
     public class FeatureLayer : Layer, IFeatureLayer  
      
    NOTE: FeatureLayer is the only example of Type Inheritance in this project.
14. Build your code and correct any errors.
15. Right click on the interfaces in the class statements to explicitly stub them out (create empty properties and methods). Do not add any code inside the { } yet!
16. Build your code and correct any errors.
17. This might be a good point to commit your code and push to GitHub.
18. In MyProGisBLL, implement the properties. That is, write code similar to the following for each property. Note that the following is code for a read-write property and not all properties in this project are read-write. Be careful with interface and property names. Use the following convention for your private member variables: an underscore followed by a camel case version of the Property name. For example:

private double \_mapScale = 0;  
   
 string IMap.MapScale

{

get

{

return \_mapScale;

}

set

{

\_mapScale = value;

}

}

NOTE: IMap in this exercise does not have a MapScale property. The above code is for illustrative purposes only.

1. Build your code and correct any errors.
2. In MyProGisBLLTests, add tests for each method in the two classes that have them.
3. You will now implement and test the methods one at a time. First all of the methods in the Map class and then all the methods in the MapDocument class. Look at the OMD on page 2. The Map class has two methods: AddLayer and RemoveLayer. Which would you implement first? Think about testing. Could you test RemoveLayer without implementing AddLayer first? Seems to make more sense to implement AddLayer first. With Test Driven Development, you will actually create a test for AddLayer first. How would you test AddLayer? In the AddLayerTest I would suggest creating a layer, giving it a name, calling AddLayer with this layer as an argument (adding the layer to \_layers), then retrieve the layer from IMap.Layers, and finally assert that the layer name that you used matches the one retrieved from IMapLayers. Create this test and run it. It should fail since you have not written any implementation code in . Have this test code and the IMap.AddLayer method code visible at the same time by using vertical or horizontal tabs. Write code in AddLayer. Run your test. Debug code in AddLayer. Run your test. Repeat until AddLayer works. Here is some starter code for AddLayer:

void IMap.AddLayer(ILayer layer)

{

// Code to resize the array and add the new layer as the last element

// in the array (similar to the last exercise’s AddLayer method)

// Array stored in \_layers private member accessed by Layers property

}

1. Write a test for IMap.RemoveLayer. Run the test and it will fail. Here is starter code for RemoveLayer.

void IMap.RemoveLayer(int index)

{

// Code to remove the Layer matching the index passed to this method

// and resize the array (similar to the last exercise’s RemoveLayer

// method)

}

1. Implement the methods of MapDocument in the following order:  
   AddMap, RemoveMap, SetFocusMap, GetMap  
   Make sure you follow the same pattern as before: one method at a time. That is, create a test for the method. Debug the method. Once it passes the test(s), move to the next method.  
   IMPORTANT NOTE: You will have to cast between interfaces to get access to the various members. For example, IMapManager.AddMap will add a map. IMapDocument.GetMap can retrieve it. This will involve code that looks a bit like the following:  
     
    IMapManager mapMgr = new MapDocument();  
    mapMgr.AddMap(map); // Assumes map was created earlier  
    … // Code to add / remove layers  
    IMapDocument mapDoc = (IMapDocument) mapMgr;  
    IMap mapCanada = mapDoc.GetMap(0);
2. Once all the tests are passing, create a Presentation Layer. That is, add a Console Application Project called MyProGisConsole to the Solution and add a reference to MyProGisBLL).  
   The console application will output the report shown below. In Program.MainWrite code to create the appropriate objects, set the property values, add the objects to the appropriate container (e.g. layers to the map), then retrieve the objects and use Console.WriteLine to write out the values to make sure they were created properly. The output should look like the following:

Map Document Report

Focus Map = U.S.

Map Name = Canada

Layer Count = 2

FeatureLayer name = Provinces

FeatureLayer featureClass = C:\data\prov.shp

FeatureLayer name = Lakes

FeatureLayer featureClass = C:\data\canlakes.shp

Map Name = U.S.

Layer Count = 3

FeatureLayer name = States

FeatureLayer featureClass = C:\data\states.shp

FeatureLayer name = Lakes

FeatureLayer featureClass = C:\data\uslakes.shp

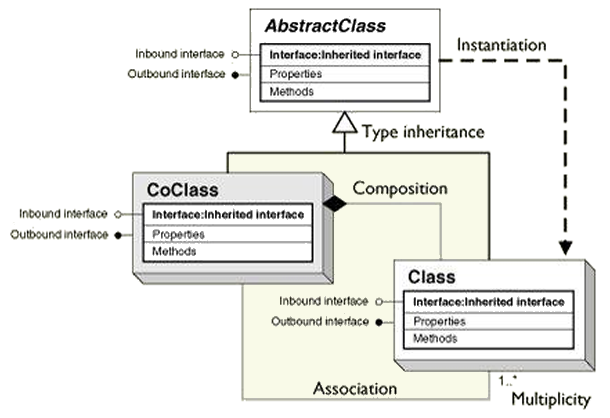
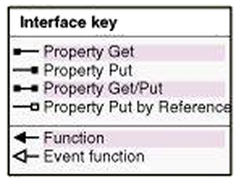
FeatureLayer name = Rivers

FeatureLayer featureClass = C:\data\usrivers.shp

1. After the lines of code required to produce the above output have been created, add calls to remove the Canada map and the Rivers layer from the U.S. map and then print a report with the same format as above.

# ESRI OMD Key

You only have Abstract and CoClass’s in this exercise.



HINT: Here is one possible implementation of RemoveLayer:

public void RemoveLayer(int index)

{

if (index < 0 || index > \_layerCount - 1)

return;

Layer[] newLayers = new Layer[\_layers.Length - 1];

for (int i = 0; i < \_layers.Length; i++)

{

if (i == index)

continue;

else if (i < index)

newLayers[i] = \_layers[i];

else

newLayers[i - 1] = \_layers[i];

}

\_layers = newLayers;

\_layerCount = \_layers.Length;

}