

# Using existing commands and adding layers

## Chapter 13 – **Executing commands**

– pp. 227-238

– **Exercise 13**

## Chapter 14 – **Adding layers to a map**

– pp. 239-261

– **Exercises 14A & 14B**

# Chapter 13 – Executing Commands

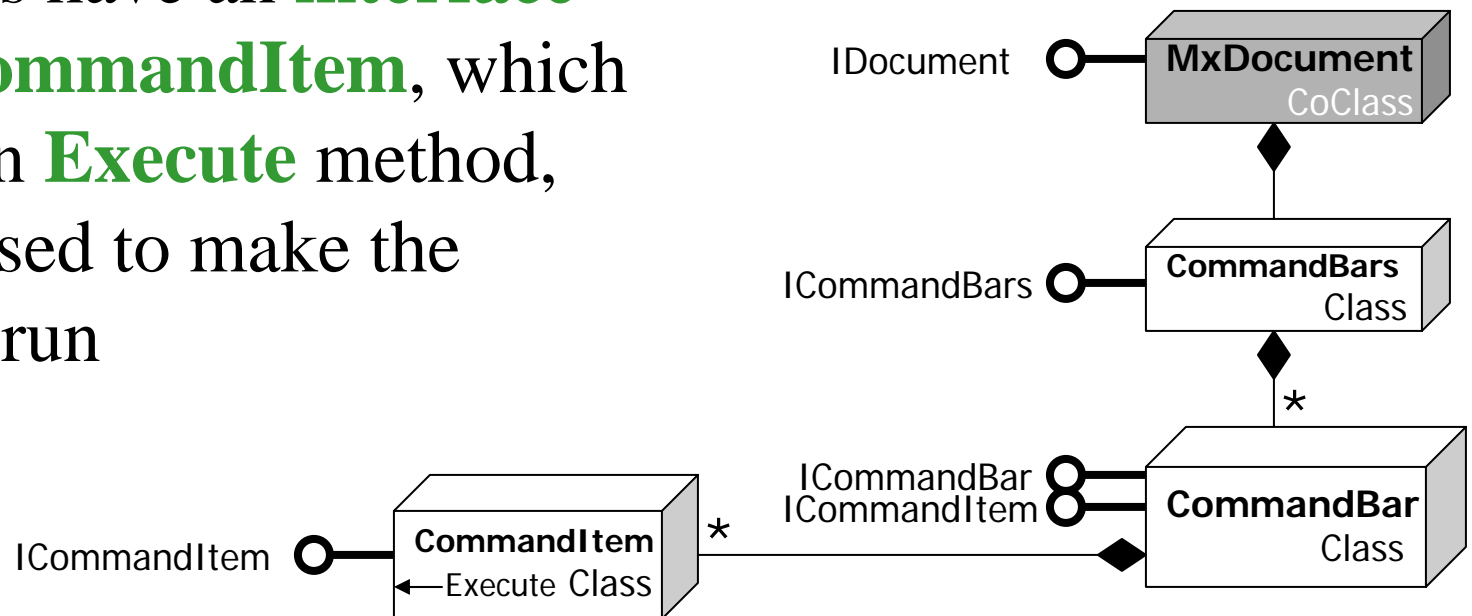
- Using CommandItems and CommandBars

# Chapter 13 – Executing Commands

- As you have seen throughout in our exploration of ArcGIS VBA, **modularity and the reusability of functionality and code** is a **key concern**
- If at all possible, we want to **avoid** reinventing the wheel:
  - If someone has already **developed the capability to perform a particular function**, the last thing we want to do is replicate their work; **we want to be able to make use of it**
- This is **equally true of ArcGIS' existing commands** and the functions they perform
  - We **do not get to see the code** that runs behind them (they are not written in VBA; using COM they were developed in C++)
  - **We can still call them**, so we can include them in our code

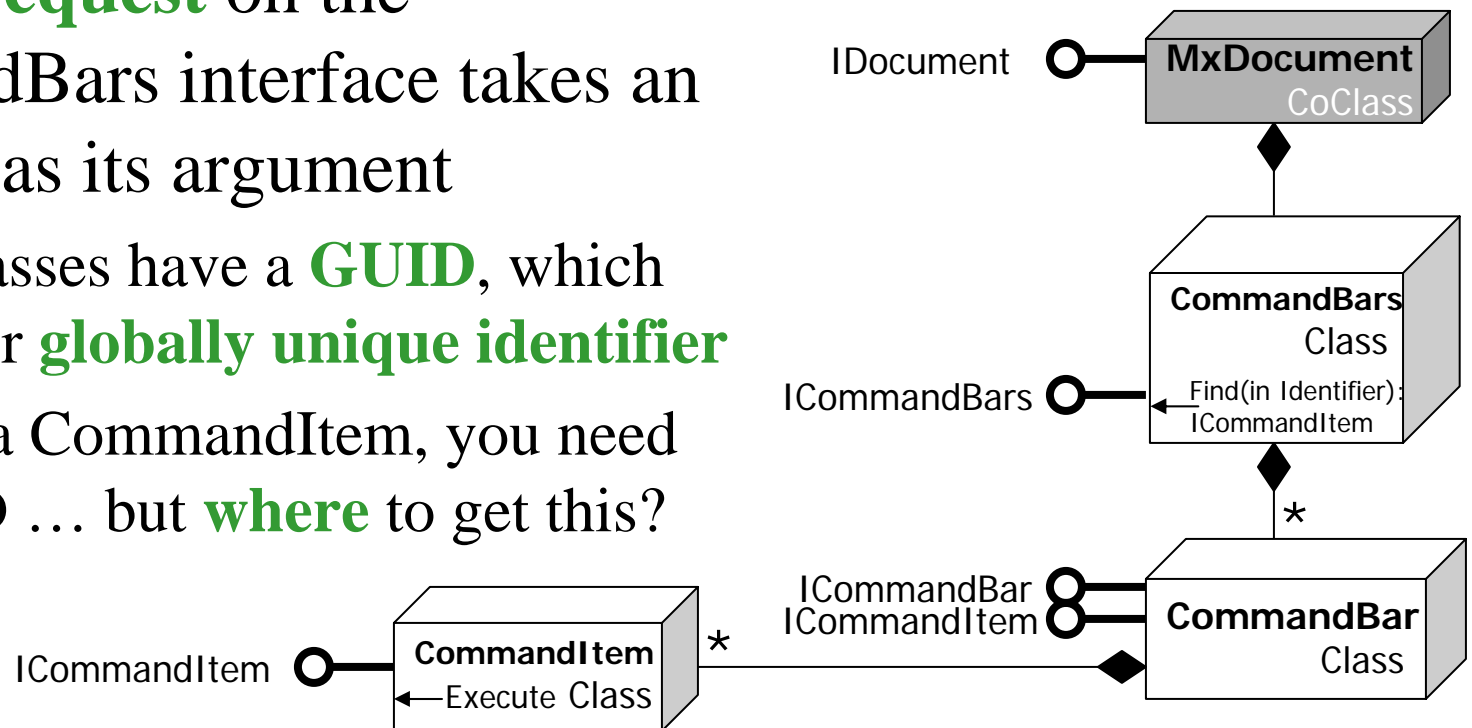
# Using CommandItems and CommandBars

- Toolbars are **composed** of commands, whether they contain tools, buttons or menu choices
  - They belong to the **CommandBar class**
  - From the notation below, you can see a **CommandBar is made up of multiple CommandItems** (commands)
- Commands have an **interface called ICommandItem**, which includes an **Execute** method, which is used to make the command run



# Using CommandItems and CommandBars

- The **CommandBars class** (note the 's' at the end) is a **collection** of all the CommandBar objects available
  - Note the **same symbology here**, showing the 'composed of multiple objects relationship'
- The **find request** on the ICommandBars interface takes an **identifier** as its argument
  - COM classes have a **GUID**, which stands for **globally unique identifier**
  - To **find** a CommandItem, you need its GUID ... but **where** to get this?



# Using CommandItems and CommandBars

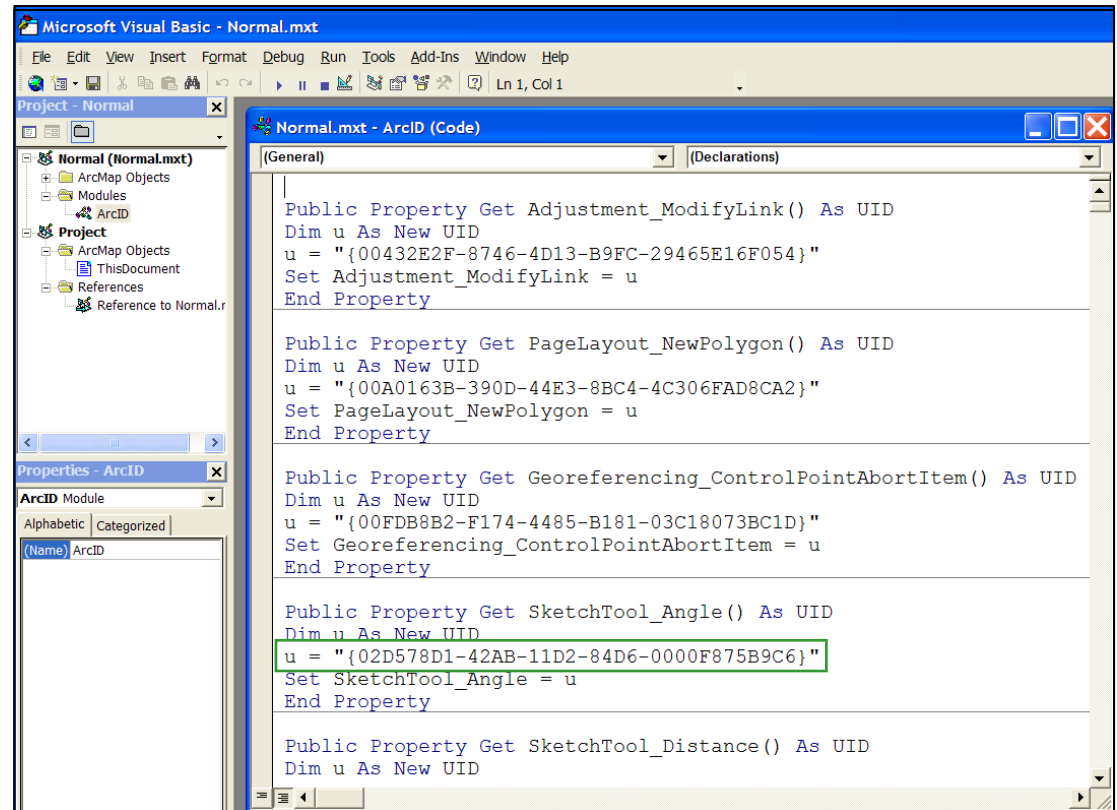
- You can **look GUIDs up in the Developer Help** in the topic *ArcMap: Names and IDs of commands and commandbars*:

The screenshot shows the ArcGIS Desktop Help for VB6 developers window. The search results for "arcmap ids" are displayed in a table. The table lists various command types, their captions, names, command categories, and their corresponding GUIDs.

Type	Caption	Name	Command Category	GUID
Toolbar	Main Menu	Main Menu	none	{1E739F59-E45F-11D0-8000-000000000000} esriArcMapUI.MxMenu
Menu	File	File_Menu	none	{56599DD3-E464-11D0-8000-000000000000} esriArcMapUI.MxFileMe
Command	New	File_New	File	{119591DB-0255-11D0-8000-000000000000} esriArcMapUI.MxFileMe
Command	Open	File_Open	File	{119591DB-0255-11D0-8000-000000000000} esriArcMapUI.MxFileMe
Command	Save	File_Save	File	{119591DB-0255-11D0-8000-000000000000} esriArcMapUI.MxFileMe
Command	Save As	File_SaveAs	File	{119591DB-0255-11D0-8000-000000000000} esriArcMapUI.MxFileMe
Command	Save A Copy	File_SaveCopyAs	File	{119591DB-0255-11D0-8000-000000000000} esriArcMapUI.MxFileMe
Command	Add Data	File_AddData	File	{E1F29C6B-4E6B-11D0-8000-000000000000} esriArcMapUI.AddData
Menu	Add Data From GIS Portal	AddInternetData_Menu	none	{5B43EFCE-8C6F-49F0-8000-000000000000} esriArcMapUI.AddInter
Command	Geography Network	IMS_ManageInternetDataURL	IMS	{C454EBA4-2DFD-49C0-8000-000000000000} esriArcMapUI.AddInter
Command	{ Add Data From GIS Portal }	{ IMS_AddInternetDataMenu }	none	{6888A697-86CB-4AC0-8000-000000000000} esriArcMapUI.AddInter
Command	Add Website	IMS_NewInternetDataURL	IMS	{C454EBA4-2DFD-49C0-8000-000000000000} esriArcMapUI.AddInter

# Using CommandItems and CommandBars

- GUIDs are **32-character hexadecimal strings**, and as such are **inconvenient to copy and paste** into code
- Instead, we can use **procedures built into the ArcID code module** of the normal.mxt project to **fetch** them
- These make it **easy to get a GUID** by getting the **appropriately named property** of ArcID



```
Public Property Get Adjustment_ModifyLink() As UID
Dim u As New UID
u = "{00432E2F-8746-4D13-B9FC-29465E16F054}"
Set Adjustment_ModifyLink = u
End Property

Public Property Get PageLayout_NewPolygon() As UID
Dim u As New UID
u = "{00A0163B-390D-44E3-8BC4-4C306FAD8CA2}"
Set PageLayout_NewPolygon = u
End Property

Public Property Get Georeferencing_ControlPointAbortItem() As UID
Dim u As New UID
u = "{00FDB8B2-F174-4485-B181-03C18073BC1D}"
Set Georeferencing_ControlPointAbortItem = u
End Property

Public Property Get SketchTool_Angle() As UID
Dim u As New UID
u = "{02D578D1-42AB-11D2-84D6-0000F875B9C6}"
Set SketchTool_Angle = u
End Property

Public Property Get SketchTool_Distance() As UID
Dim u As New UID
```

ArcID.SketchTool\_Angle

# Using CommandItems and CommandBars

- Putting this **all together**:

```
Dim pCommandItem As ICommandItem
Set pCommandItem = CommandBars.Find(ArcID.SketchTool_Angle)
pCommandItem.Execute
```

- Getting a **toolbar** works in a **similar fashion**

– **Toolbars have GUIDs** too, and can be found in the same way

```
Dim pCommandItem As ICommandItem
Set pCommandItem = CommandBars.Find(ArcID.Editor_EditorToolbar)
```

- However, toolbar properties and methods are on the ICommandBar interface (not ICommandItem), so we QueryInterface to get the right interface:

```
Dim pCommandBar As ICommandBar
Set pCommandBar = pCommandItem
```



# Chapter 14 – Adding layers to a map

- Adding a geodatabase feature class
- Adding a raster data set

# Chapter 14 – Adding layers to a map

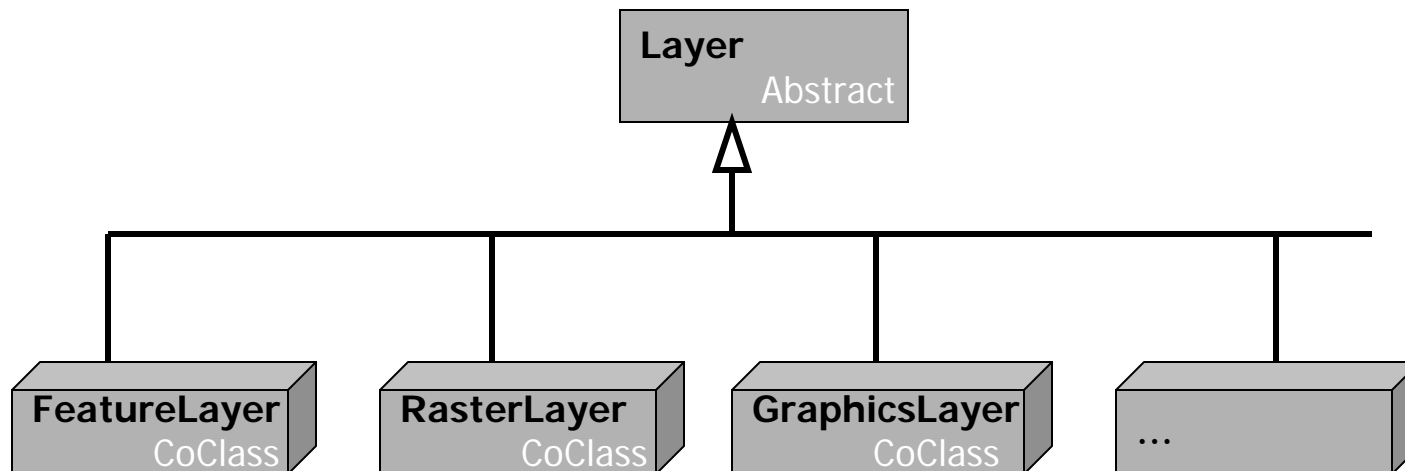
- **Adding layers to maps through the GUI** is something every user does when they use ArcMap
- Equally important to the developer is **to be able to add layers using code**, as this is a necessary precondition to doing something to the layers with the code
- This is really a **four step** process:
  1. **Create the layer** from one of the layer coclasses
  2. **Get the data set from a storage location** that the computer can access (either locally or somewhere networked)
  3. **Associate** the data set with the layer
  4. **Add the layer** to the map

# Chapter 14 – Adding layers to a map

- The first step, creating the layer from one of the layer coclasses, uses **straightforward VBA code**:

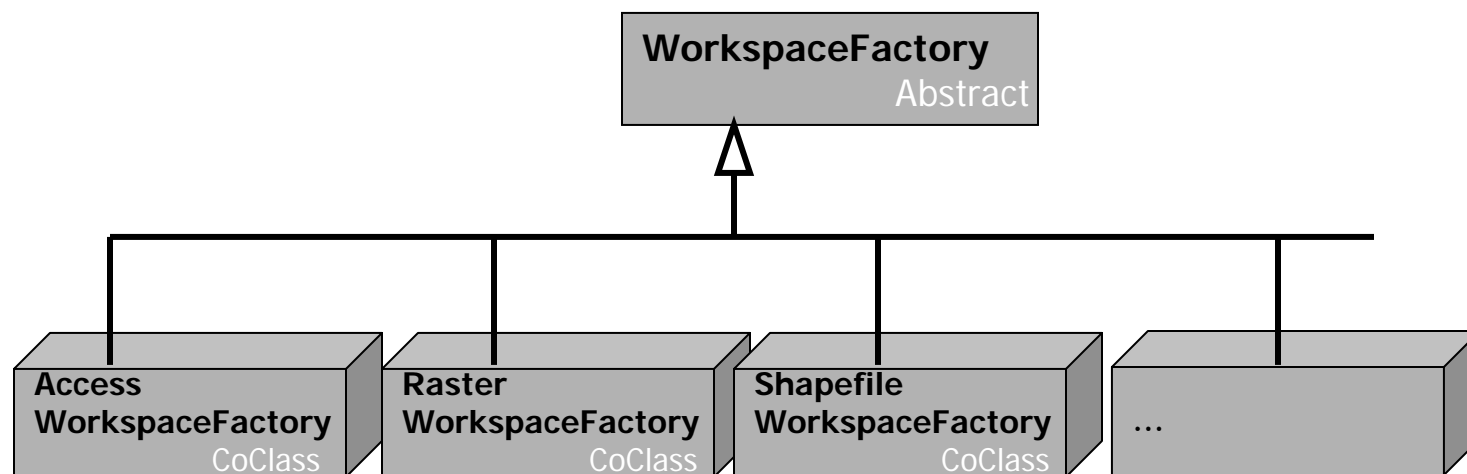
```
Dim pRLayer as IRasterLayer  
Set pRLayer = New RasterLayer
```

- The key is to **identify the appropriate type** of layer:



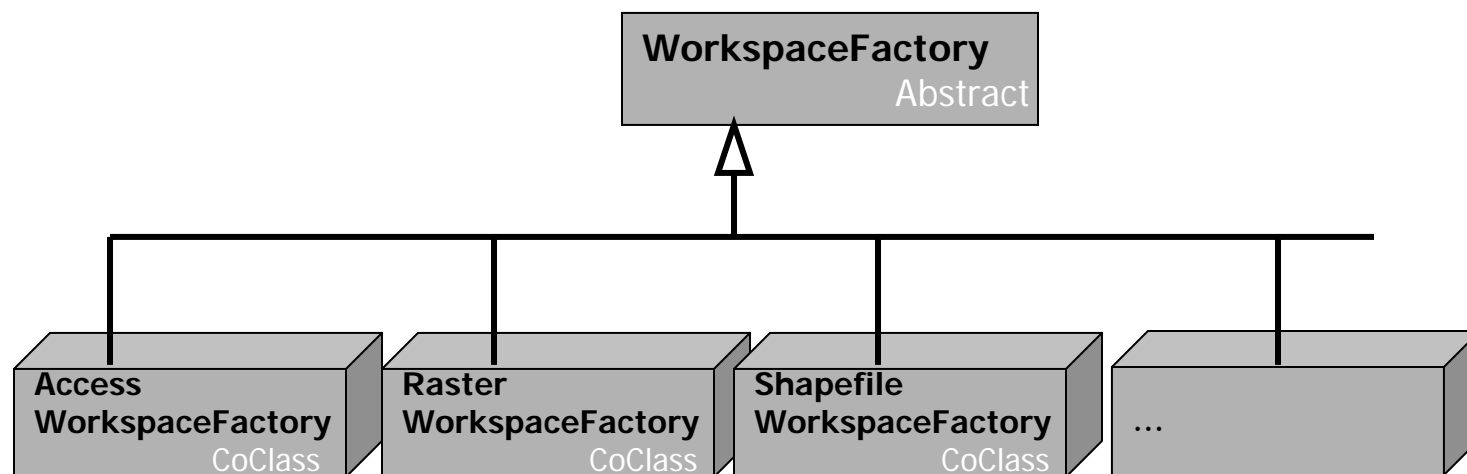
# Chapter 14 – Adding layers to a map

- The second step, **getting the data set**, is a little **more tricky** ... partly because ArcGIS is **so flexible** with data
  - Because ArcGIS can work with **so many different kinds of data files**, there are **lots of variations** on this
- To simplify the process, in all cases to get a data set, one must first **get its workspace**, which one **creates using a workspace factory**:



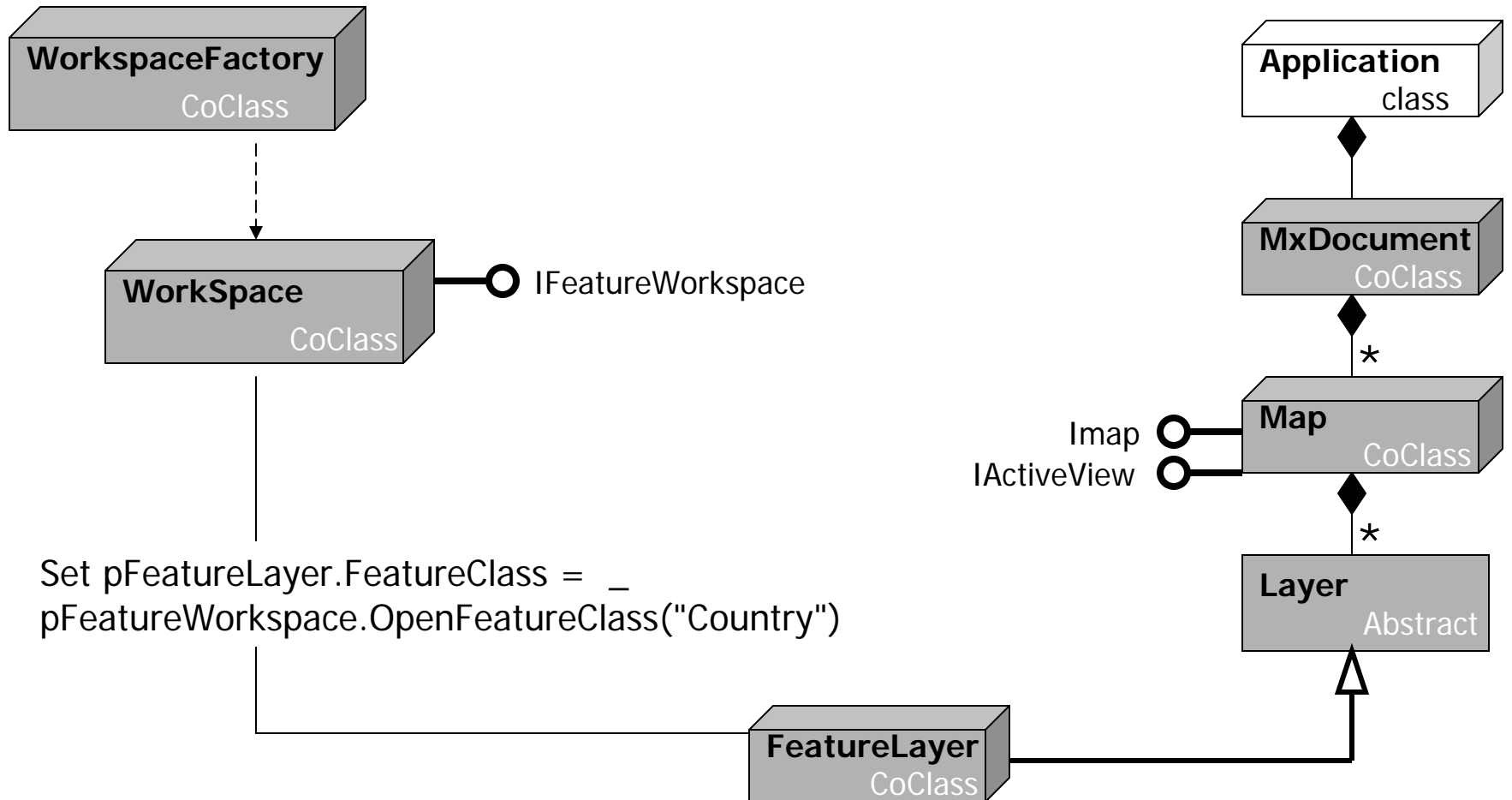
# Chapter 14 – Adding layers to a map

- You **select** the right WorkspaceFactory from the many coclasses, and use it to **create the required workspace**
- **Workspaces are composed of data sets** (which is what we are really after)
- There are WorkspaceFactories **specific to each type of data set files** we might want to add to our map:



# Chapter 14 – Adding layers to a map

## ShapeFile Example



# Adding a geodatabase feature class

- Your first exercise will take you through the **four step process** using a **geodatabase feature class**
- The first key thing that **you need to know**, both here and in all cases really, is the **kind of data file** in question → this **determines the right kind of WorkspaceFactory**
- Here we are working with an **MS Access database**, so we need an **AccessWorkspaceFactory**:

```
Dim pAWFactory As IWorkspaceFactory  
Set pAWFactory = New AccessWorkspaceFactory
```

- The IWorkspaceFactory interface has an **OpenFromFile method** that is used to open the file:

```
Dim pFWorkspace As IFeatureWorkspace  
Set pFWorkspace = pAWFactory.OpenFromFile("thefile.mdb", 0)
```

# Adding a geodatabase feature class

- We now have the Workspace required and we can now **get the feature class** with the **OpenFeatureClass** method on the IFeatureWorkspace interface of our Workspace:

```
Dim pFClass As IFeatureClass  
Set pFClass = pFWorkspace.OpenFeatureClass( "Roads" )
```

- Setting up a feature layer and associating it with the class is relatively **straightforward**:

```
Dim pFLayer As IFeatureLayer  
Set pFLayer = New FeatureLayer  
Set pFLayer.FeatureClass = pFClass
```

- Finally, adding it to the Map document is equally **straightforward** (see the text for the five lines of code required)



# Adding a raster data set

- Your second exercise involves a **similar procedure**, only this time the data set is **raster data rather than features** from within a geodatabase
- The only real wrinkle is switching to **use the right WorkspaceFactory for the particular kind of data ...** but the hope is that once you have done this for two different sorts of data, you will be **comfortable** with doing it for **any sort of data set**
- This way, you will have worked with **data sets from both the vector and raster spatial data models**, which covers most of what you are likely to work with in real applications

# **Next Topic:**

Map symbology and ArcCatalog