Working with layouts and editing data

Chapter 19 – **Making dynamic layouts**

- pp. 359-376
- **Exercises 19A & 19B**

Chapter 20 – **Editing tables**

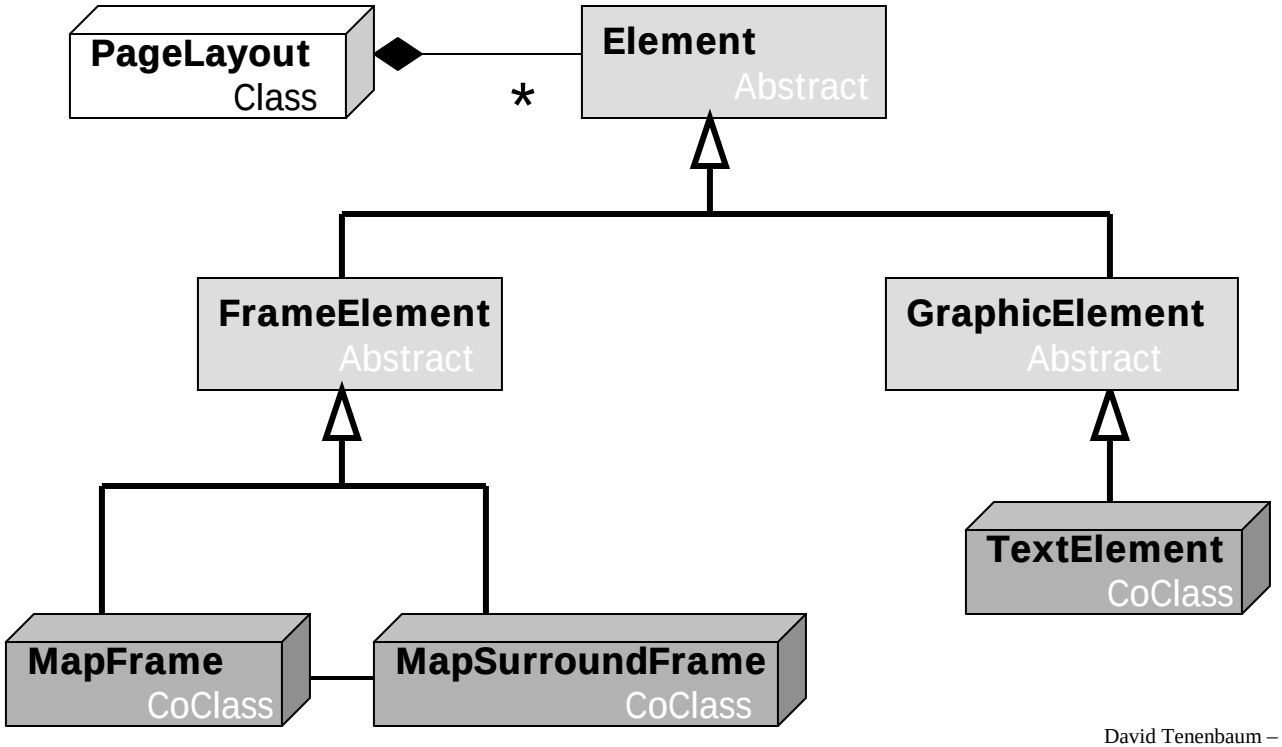
- pp. 377-401
- **Exercises 20A & 20B**

Chapter 19 – Making dynamic layouts

- Naming elements
- Manipulating text elements

Chapter 19 – Making dynamic layouts

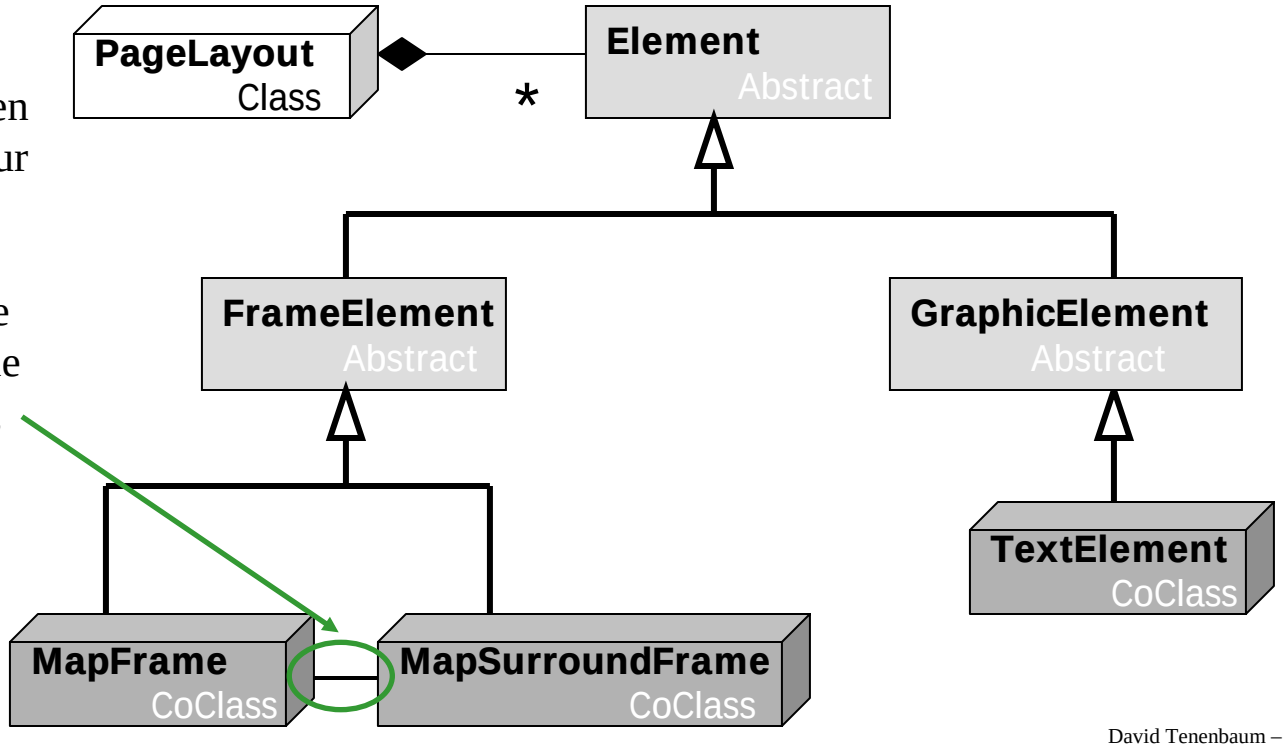
- All the **items found in a map layouts** are, within VBA, objects known as **Elements**
  - The Element class is an **abstract class**, which forms of the basis of **several types of elements** (we used GraphicElements in our Chapter 12 exercises):



Chapter 19 – Making dynamic layouts

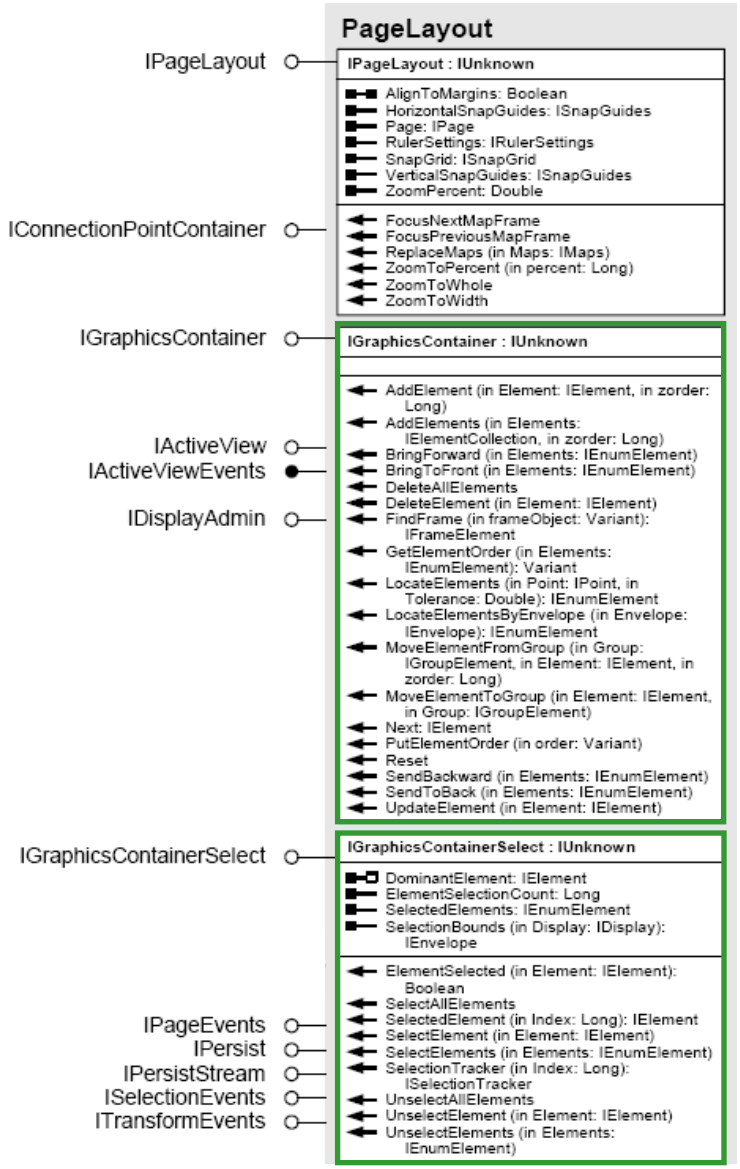
- The split between **FrameElement** and **GraphicElement** is **important**, because they each **behave differently**:
  - **FrameElements** (like data frames and their associated elements) **update to reflect any changes in the map** shown; On the other hand, **GraphicElements do not** … normally they are **static**

This linkage between these coclasses is our indication of their relationship; the MapSurroundFrame will update when the MapFrame updates, for example.



Chapter 19 – Making dynamic layouts

- We will **write code** to make the **static GraphicElements** in a layout **change in response to the changes a user makes** in the FrameElements
- When we work with the **set of elements in a layout**, it is made easier by the fact that we have some **interfaces** on the **PageLayout class** that make it easier to work with a **collection of elements**:

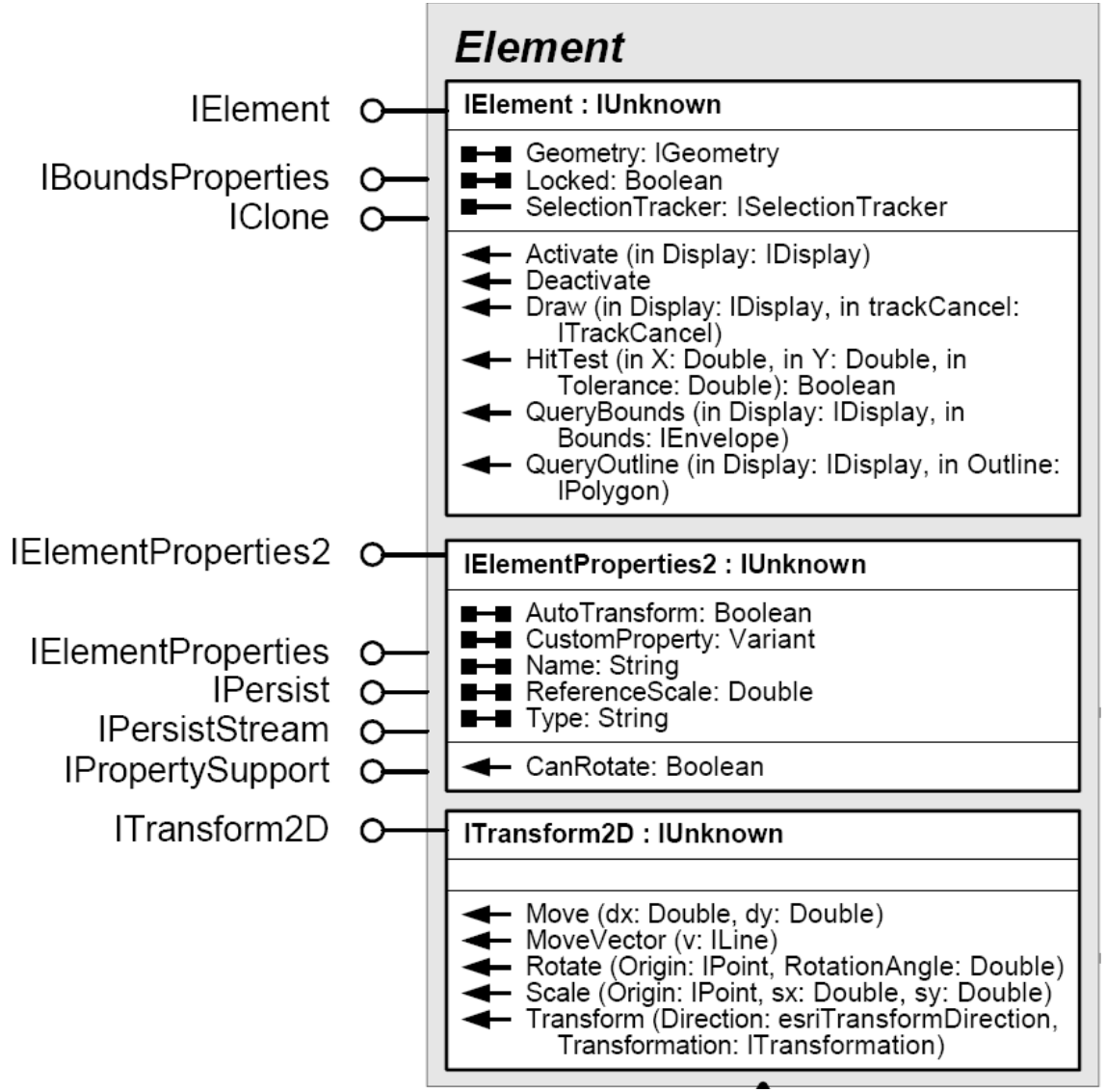


Chapter 19 – Making dynamic layouts

- Both **IGraphicsContainer** & **IGraphicsContainerSelect** can be used to **collect all kinds of elements** (Frame and Graphic alike)
- You can use **IGraphicsContainer** to add, delete and reorder elements (in ways similar to other collection objects we have used before)
- The **IGraphicsContainerSelect** interface provides a method (**DominantElement**) by which you can get the **currently selected element(s)**, and also provides an **ElementSelectionCount** property to get the **current number of selected elements**

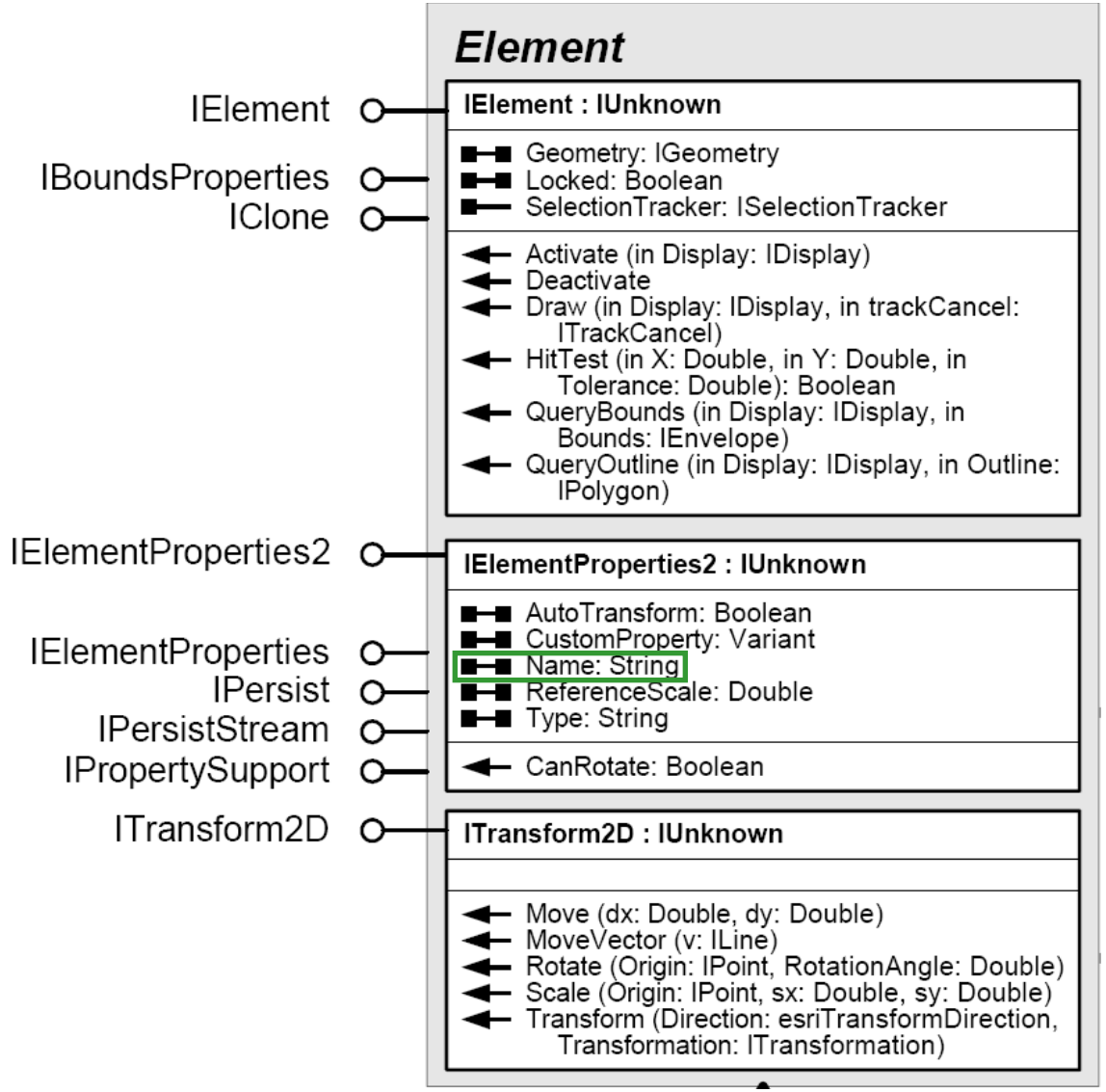
Naming elements

- In this chapter's exercises, you will **change the text elements** in your layout based on some of the code you have developed in previous chapters
- This involves **finding the right elements**, and **updating their properties** according to choices the user makes
- The **tricky part** of this is **identifying the elements you need to change**; this is **easy visually**, but **hard** to do **by code**



Naming elements

- **IElementProperties2** provides a **Name property**
  - Once this has been set, we have an **easy way to find an particular element** within the graphics container
- We will **create buttons to let us get and set element names** to make this convenient for the user



Manipulating text elements

- Once we have got the **functionality set up to get and set our elements' names**, we will make use of it
- We will **use the Name property to find particular elements** by checking through each of the elements that is present in the graphics container to find the right one (based on the name matching)
- We begin by **getting the graphics container** we need, letting VBA do an automatic QueryInterface for us:

```
Dim pMxDoc As IMxDocument
Set pMxDoc = ThisDocument
Dim pGraphics As IGraphicsContainer
Set pGraphics = pMxDoc.PageLayout
```

Manipulating text elements

- We can now **get elements from the graphics container** sequentially using its **Next method**:
  - The Next method returns the **IElement interface** of the element it gets, but we can **use an automatic QueryInterface** to get the interface we really want (**IElementProperties2**, that has the **Name property** on it):

```
Dim pElementProp As IElementProperties2
Set pElementProp = pGraphics.Next
```

- Each time we **get the next element**, we can then **check its name against what we are looking for** using an **If Then (or Case) statement**:

```
If pElementProp.Name = "ToxicMapTitle" Then
```

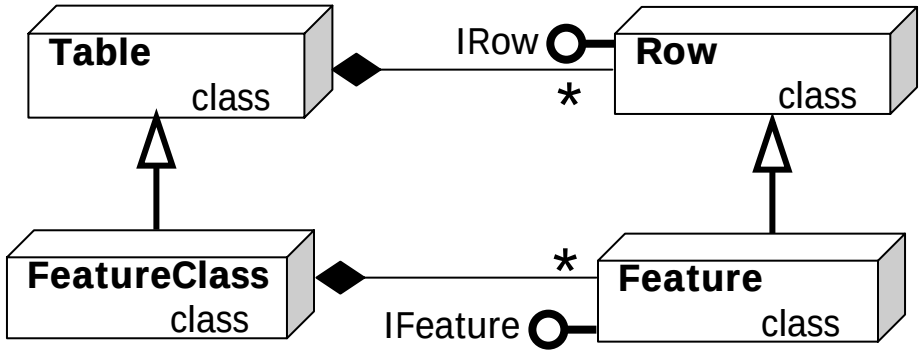
- Once we **find the right one**, we can **set its Text property**

Chapter 20 – Editing tables

- Adding fields
- Getting and setting values

Chapter 20 – Editing tables

- Recall that the **features** we work with in ArcGIS are actually **stored as records in a** table:



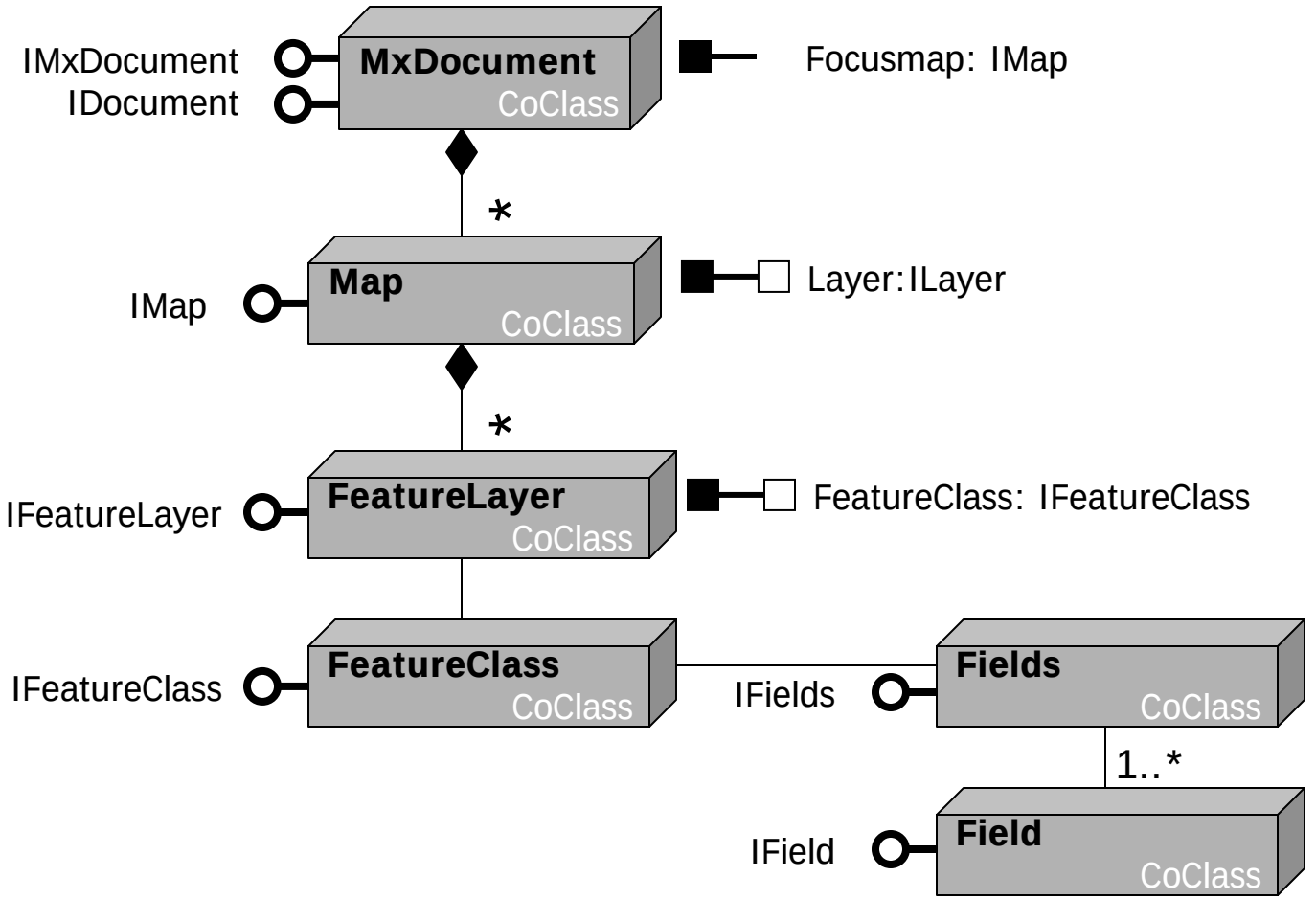
- Tables have a **second dimension** as well: **Columns** in the table represent categories of information. These are **actually stored as fields** in a table
- The **intersection** of a record and a field is a cell; this holds a particular **piece of information known as a value**

Chapter 20 – Editing tables

- There are two ways we will **modify tables**:
  1. We will **add fields to tables** to increase the number of categories of information we can store in them
  2. We will **edit cell values stored in the table**
- These are absolutely **key skills** for an ArcGIS VBA coder: Once you write your custom application to do some spatial analysis, you will **need to be able to store the results!**
- Editing cell values will make use of **cursors** (which we worked with in Chapter 18)
  - We will make cursors, move their pointer to a particular record, and specify a particular field to **specify the cell of interest**

Adding fields

- A **feature class** has a **Fields** object, which is a **collection** comprised of **all of its Field** objects:



Adding fields

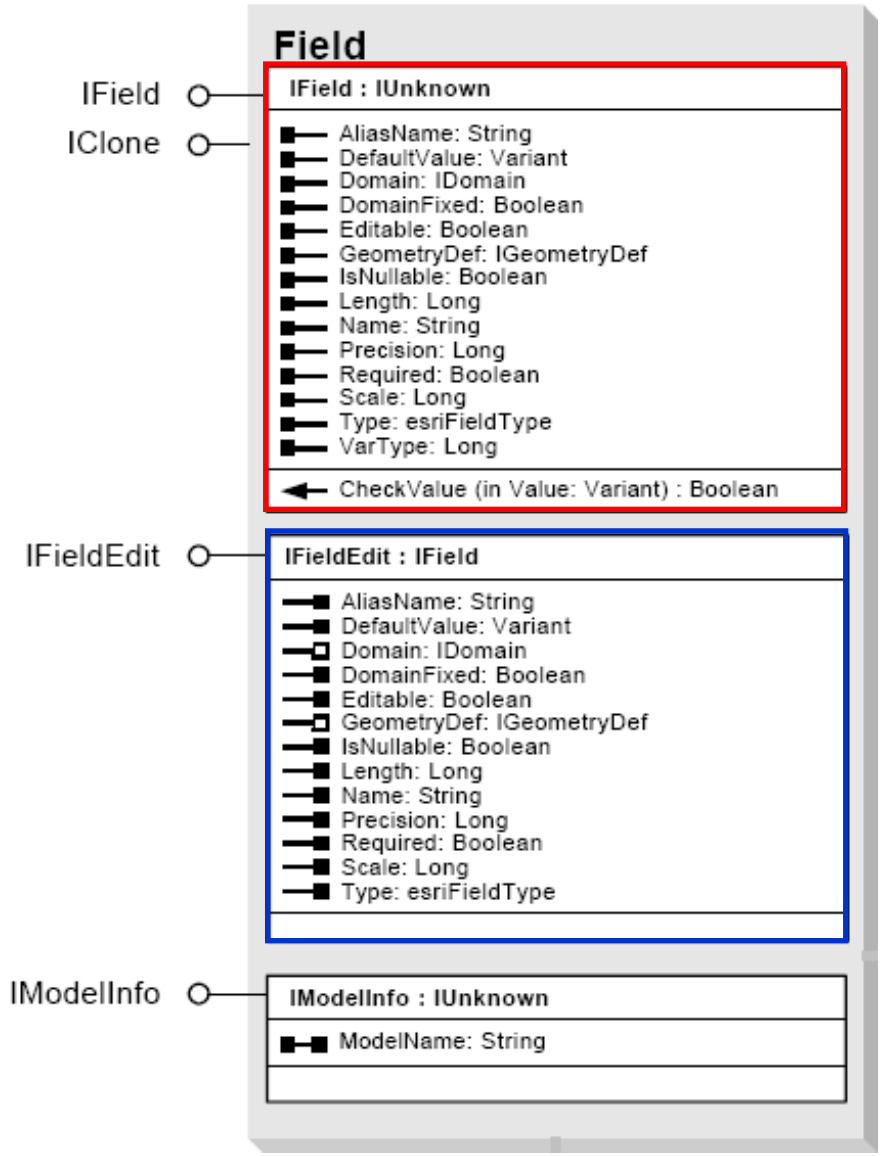
- To **add a field** to a feature class, you first have to **make a new field** from the **Field** coclass in the usual fashion:

```
Dim pField As IField
Set pField = New Field
```

- Once you have **created the field**, you must **set its properties** to make it suitable for the kind of information you want to store within it
  - This needs to be done **before it gets added to table**; once added to a table you can **no longer change the Field's properties**
- The Field coclass has **two nearly identical interfaces**, with **one key difference** between them: One is solely designed for **getting properties**, and the other is for **setting them**

Adding fields

- The **IField interface** has **only left-hand barbells**, so it can only be used to **get a Field's properties**, but not set them
- The **IFieldEdit interface** has **only right-hand barbells**, so it can only be used to **set a Field's properties**, but not get them
- **BUT** … **IFieldEdit inherits from IField**, so … (What does this mean?)



Adding fields

- Once you have made a field, **use the IFieldEdit interface to set its properties**, either by having declared a variable to that interface initially, or by switching to it now:

```
Dim pFieldField As IFieldEdit
Set pFieldEdit = pField
```

- The **two properties** of a Field that you will **always need to set** are its **name** (which is a string) and the **data type**:

```
pFieldField.Name = “Population”
pFieldEdit.Type = esriFieldTypeInteger
```

- There are a number of **field types**, and you can look these up in the help to see which of them you would want to use for a **particular kind of information**

Adding fields

- **Before adding a new field** to a table, it is safest to first **check** and make sure that **table doesn't already contain a field** with the **same name**
  - This is where the **Fields** object comes in handy, because if has a **FindField request** that will look for a field with a certain name:

```
Dim pFields As IFields
Set pFields = pFClass.Fields
Dim intPosPopField as Integer
intPosPopField = pFields.FindField("Population")
```

- The FindField request **returns the index** of a **Field object's position** in the Fields collection, with the **first field denoted by the value 0**
  - If FindField **does not find a field with the specified name**, it **returns the value -1**

Adding fields

- We can now **use an If Then statement to check** if the field is **present or not**, and proceed accordingly

```
If intPosPopField = -1 Then End If
```

- Supposing we do not find a field with a matching name (because FindField does return -1), we can then **add our new field using the AddField method** of IFeatureClass:

```
pFClass.AddField pField
```

- In Exercise 20A, we will develop code to **add a field to a specific feature class**
  - We will put the button for this in an **unusual place**: We will place it on the **feature layer context menu** (the one you get when you right-click a feature layer in the Table of Contents), and only when **the right kind of layer** (feature) is clicked upon

Adding fields

- The **required functionality** for pulling up the feature layer context menu when you right-click a feature layer in the Table of Contents is accomplished through the **MxDocument** object's **ContextItem property**
- A user can **right-click on lots of different things** in the GUI:
  - This is why the **ContextItem property returns the IUnknown interface**; lots of kinds of objects inherit this interface:

```
Dim pMxDoc as IMxDocument
Set pMxDoc = ThisDocument
Dim pUnknown As IUnknown
Set pUnknown = pMxDoc.ContextItem
```

Adding fields

- In the case of our code here, **we know that the user has right-clicked on a feature layer in the TOC**, because they can **ONLY** **get to our button** that runs this code if they have done so
- Thus, **we know what has been returned** by the **ContextItem** property is **actually a feature layer**:

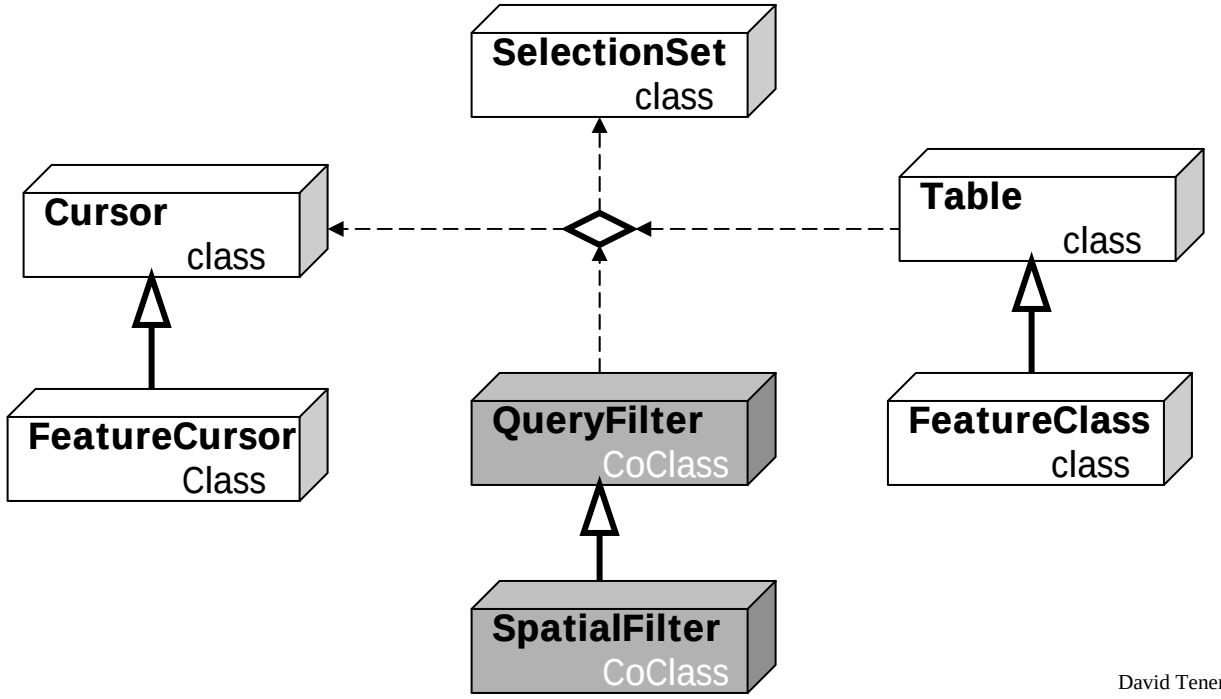
```
Dim pFLayer as IFeatureLayer
Set pFLayer = pUnknown
```

- Finally, we can **get the feature class from the feature layer** that was returned, so we can now **proceed to add our desired field**:

```
Dim pFClass as IFeatureClass
Set pFClass = pFLayer.FeatureClass
```

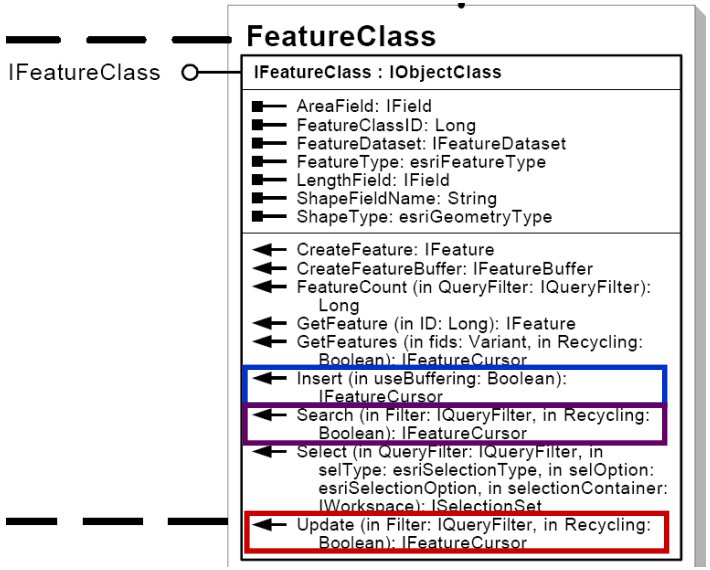
Getting and setting values

- In Exercise 20B, you will accomplish **two tasks**:
  1. You will **cycle through all the records**, examining the values in a field, & **replacing all Null values with calculated values**
  2. You will cycle through the values again and **sum them up**
  - To go through all the records in a feature class, you will make a **feature cursor** (recall them from Chapter 18):



Getting and setting values

- Recall that the **IFeatureClass interface** (which we used to make selection sets) has **three methods** to **make a feature cursor**:



1. The **Insert method** lets you **add new features** to a feature class
2. The **Update method** lets you **edit existing features**
3. The **Search method** makes a cursor that **contains all features satisfying a query statement**
   - This is useful when you want to **get information** about features but **do not want to make any new features**

Getting and setting values

- This time, we will make a feature cursor using the **Update method**, and we **do not need to make a query filter** because we want to **get all the records** (rather than a subset of them):

```
Dim pFCursor As IFeatureCursor
Set pFCursor = pFClass.Update(Nothing, False)
```

- We can now move through the records one at a time with the feature cursor's **NextFeature method**:

```
Dim pFeature As IFeature
Set pFeature = pfCursor.NextFeature
```

- The NextFeature method can be **repeated** until the pointer is **pointing at the desired feature**

Getting and setting values

- Once you have the right feature, you can use the **Value property** on the **IRowBuffer interface** to **get or set the value** for a field denoted by an index value, e.g.

  ```
  pFeature.Value(3) = 60000
  ```

  will set the value in the 4th field (remember, the first has index = 0) for the record of interest to 60000
- Once this is done, you have **changed that value in memory**; to make this a **permanent change recorded in the file** corresponding to the table, use the feature cursor's **UpdateFeature method**:

  ```
  pFCursor.UpdateFeature pFeature
  ```
- Use a **Do Until** loop to **change all features** in the cursor

EEOS 472 – Programming for GIScience Applications

- Students will be provided with **hands-on experience**, working with Visual Basic for Applications (VBA), which is integrated into the ArcGIS desktop geographic information system (GIS). The **goals** are to help students:

1. Understand the key concepts of **object-oriented programming**
2. Become skilled at **using VBA** to customize ArcGIS
3. Build capability and understanding in the **application of programming techniques** to GIScience applications

Next Topic:

Final review and final exam