

The object model, UML diagrams, and making tools

Chapter 11 – Navigating object model diagrams

- pp. 171-197
- Exercises 11A & 11B

Chapter 12 – Making tools

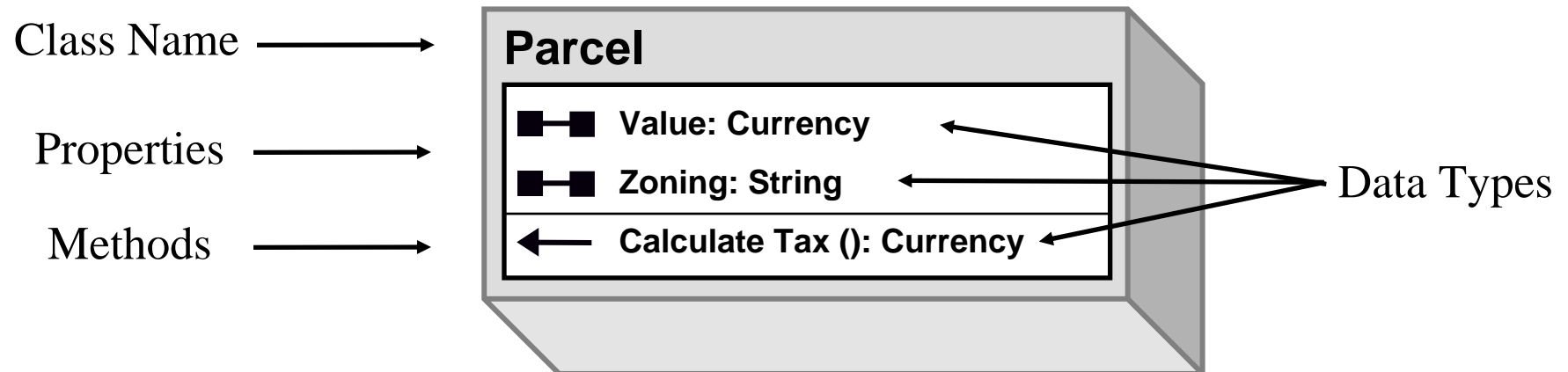
- pp. 199-225
- Exercises 12A, 12B & 12C

Chapter 11 – Navigating object model diagrams

- Getting layers
- Creating and assigning colors

Chapter 11 – Navigating object model diagrams

- In our previous lecture, we introduced **Unified Modeling Language (UML) diagrams** for classes:



- While these diagrams are useful to us just to see the **characteristics of a class**, their **real power** comes in showing us the **relationships between classes** in the **ArcGIS object model**

Chapter 11 – Navigating object model diagrams

- The **following slides** appeared in a slideshow entitled *Getting Started with ArcObjects*, presented at the 2009 ESRI User Conference held on July 13 – 17, 2009 in San Diego, CA in Technical Workshops lead by Toni Fisher, Ken Smith and Patrisha Wells:

http://proceedings.esri.com/dvd/uc/2009/uc/tws/workshops/tw_195.pdf

- I've included their **Diagrams section** here because it does an excellent job of illustrating how the ArcGIS object model diagrams **make use of UML**
- I little Googling leads me to believe that a lot of this material was originally **developed by Robert Burke** (the author of our textbook)

Lecture Path

What are
ArcObjects

GIS
parts

Diagrams

**Interfaces
& COM**

**Implement
ESRI
Interfaces**

**Extend
the
Applications**

Questions

Wrap-up

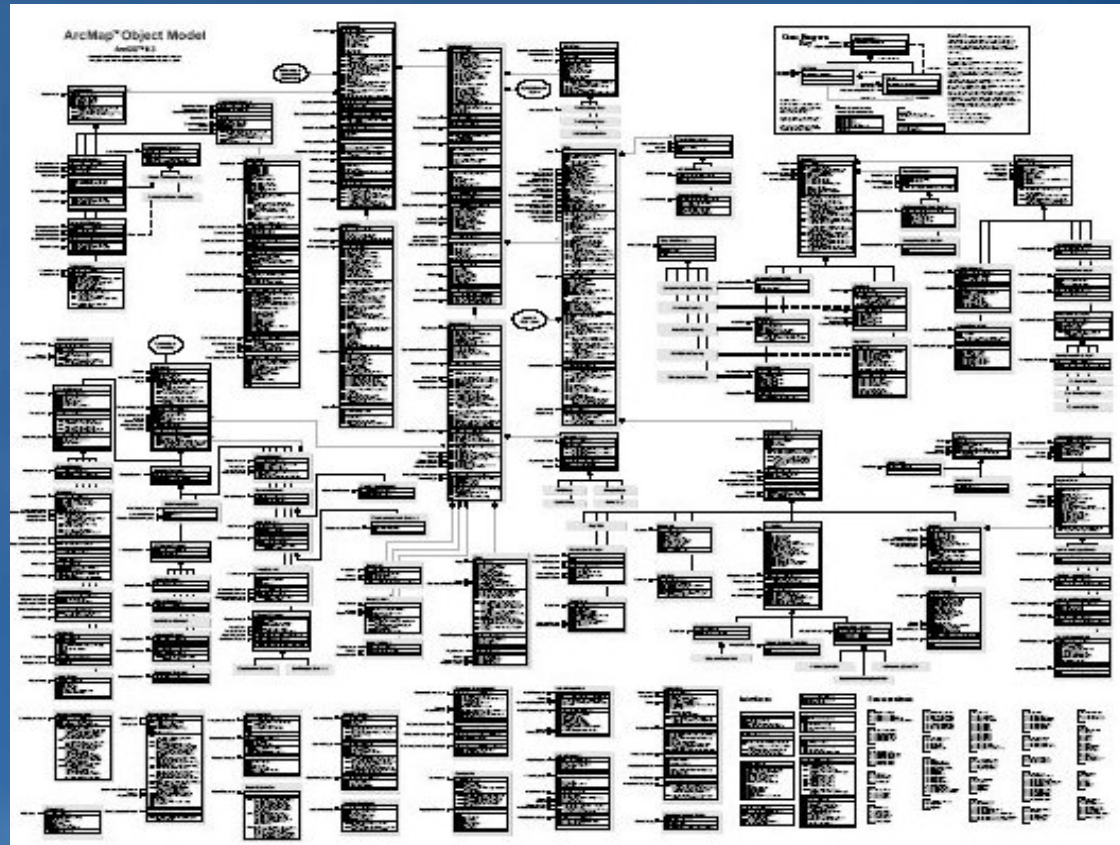
Engine

Server

**Extend
the
Geodatabase**

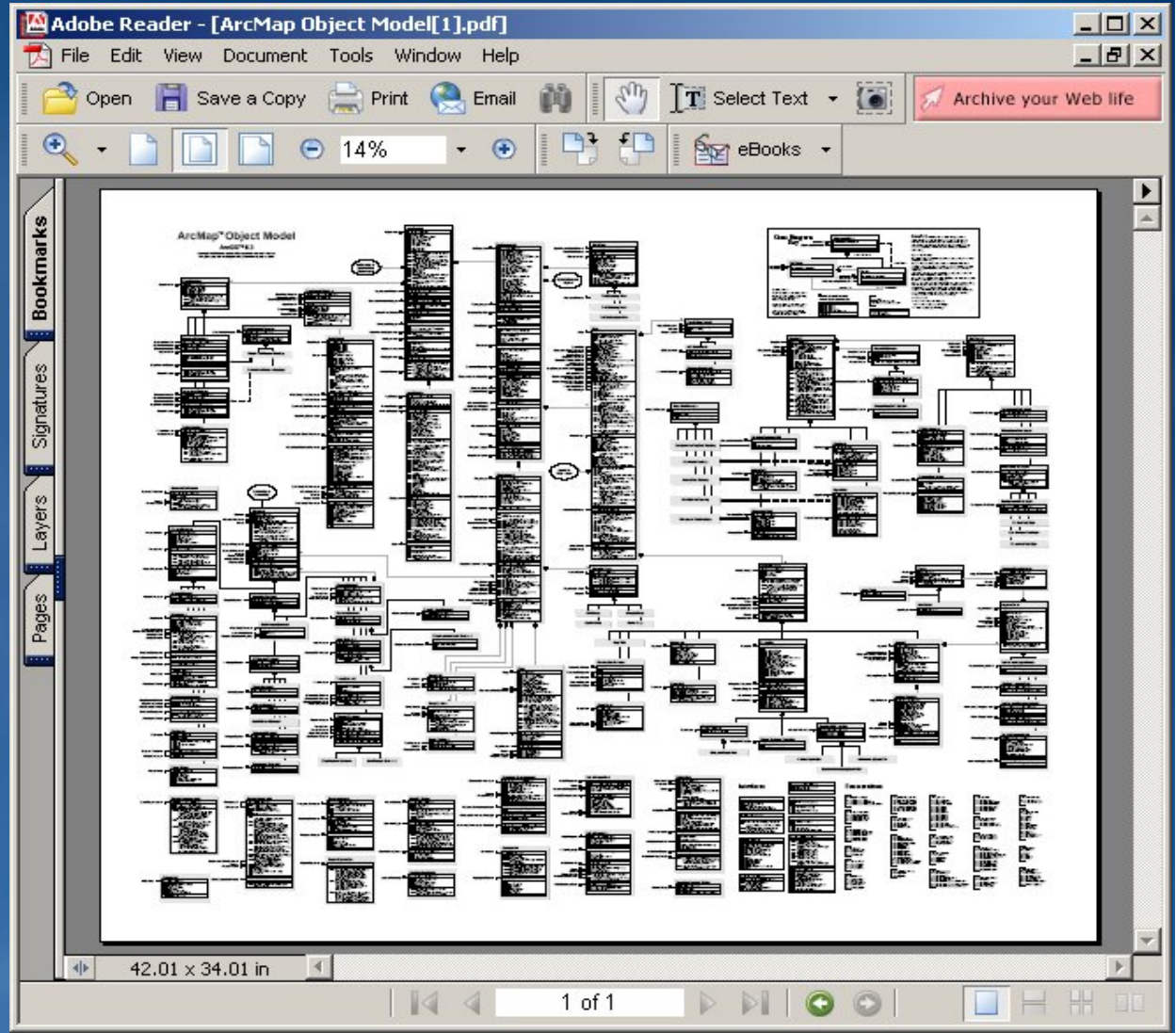
Object Model Diagrams

- Road maps to the ArcObjects classes
- Help you write code
- Based on UML – Unified Modeling Language
 - Symbols show relationships, connections, properties, and methods
- > 70 libraries
- > 110 posters



Where do you get the diagrams?

- Install the ArcObjects developer kit
 - C:\Program Files\ArcGIS\DeveloperKit\Diagrams*PDF
- Help
 - Per library
- Online
 - ESRI Resources site
 - Per library



Navigating the help: Carto library, diagram, and classes

The screenshot shows a Microsoft Document Explorer window displaying the help content for the Carto library. The window title is "ms-help://ESRI.EDNv9.3/esriCarto/CartoObjectModel.pdf - EDN_Header - Microsoft Document Explorer". The main content area shows a class diagram titled "Carto Object Model Map Elements". The left pane shows the "Contents" view, which is filtered by "(no filter)". The tree structure is as follows:

- Building solutions with ArcGIS Engine using .NET
 - What's new at 9.3?
 - Tips for using the installed ArcGIS .NET SDKs
 - Query for topics about...
 - Getting started
 - Visual Studio Integration Tools
 - Programming with .NET
 - Working with ArcGIS components
 - Licensing your solution
 - Deploying your solution
 - Tools
 - General reference
 - ArcObjects library reference
 - Understanding the ArcObjects library reference
 - ESRI.ArcGIS.ADF
 - ESRI.ArcGIS.ADF.Connection
 - 3DAnalyst
 - Animation
 - ArcWeb
 - Carto
 - Overview
 - ESRI.ArcGIS.Carto
 - Carto Namespace Contents
 - Carto Namespace Object Model Diagram
 - Interfaces
 - Classes

Four orange arrows point to the "Building solutions with ArcGIS Engine using .NET" section, the "ArcObjects library reference" section, the "Carto" section, and the "Carto Namespace Object Model Diagram" section. A red arrow points to the "Carto Namespace Object Model Diagram" section.

Carto's Map class with lots of symbols

The symbols help you write code

The screenshot displays a software interface with a class hierarchy on the left and a detailed view of the 'Map' class on the right. The interface includes a navigation bar at the top with a back arrow, a forward arrow, a page indicator '1 / 11', a zoom level of '100%', and a search icon. The class hierarchy on the left lists various classes, with 'IMap' selected. The detailed view of the 'Map' class on the right shows its inheritance from 'IMap : IUnknown' and lists its properties and methods. A diamond symbol on the right side of the 'Map' class view indicates its inheritance relationship.

Map

IMap : IUnknown

- ActiveGraphicsLayer: ILayer
- AnnotationEngine: IAnnotateMap
- AreaOfInterest: IEnvelope
- Barriers (pExtent: IEnvelope): IBarrierCollection
- BasicGraphicsLayer: IGraphicsLayer
- ClipBorder: IBorder
- ClipGeometry: IGeometry
- Description: String
- DistanceUnits: esriUnits
- Expanded: Boolean
- FeatureSelection: ISelection
- IsFramed: Boolean
- Layer (in Index: Long): ILayer
- LayerCount: Long
- Layers (in uid: IUID, in recursive: Boolean): IEnumLayer
- MapScale: Double
- MapSurround (in Index: Long): IMapSurround
- MapSurroundCount: Long
- MapUnits: esriUnits
- Name: String
- ReferenceScale: Double
- SelectionCount: Long
- SpatialReference: ISpatialReference
- SpatialReferenceLocked: Boolean
- UseSymbolLevels: Boolean

Methods:

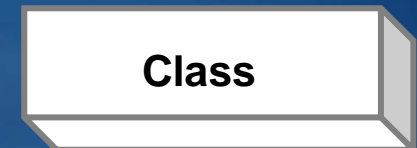
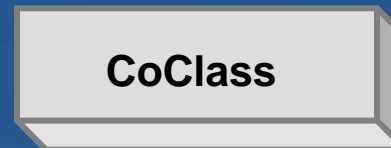
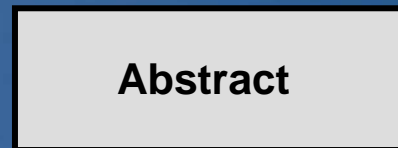
- AddLayer (in Layer: ILayer)
- AddLayers (in Layers: IEnumLayer, in autoArrange: Boolean)
- AddMapSurround (in MapSurround: IMapSurround)
- ClearLayers
- ClearMapSurrounds
- ClearSelection
- ComputeDistance (in p1: IPoint, in p2: IPoint): Double
- CreateMapSurround (in clsid: IUID, in optionalStyle: IMapSurround): IMapSurround

The twelve UML Symbols

- Relationships



- Classes



- Properties and methods



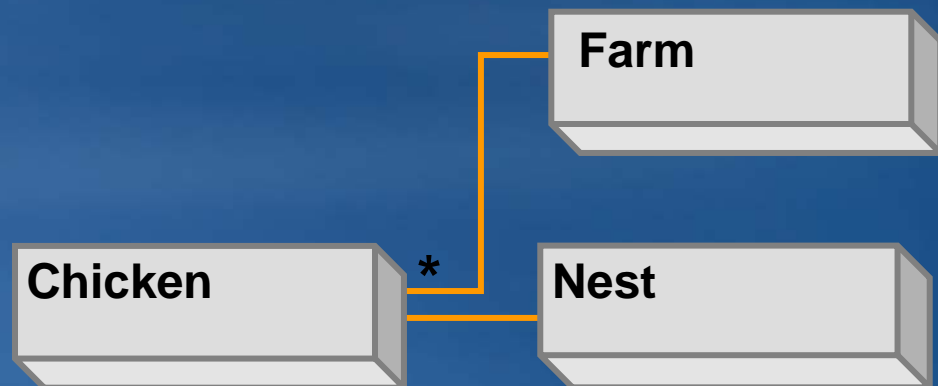
UML symbols

- Association



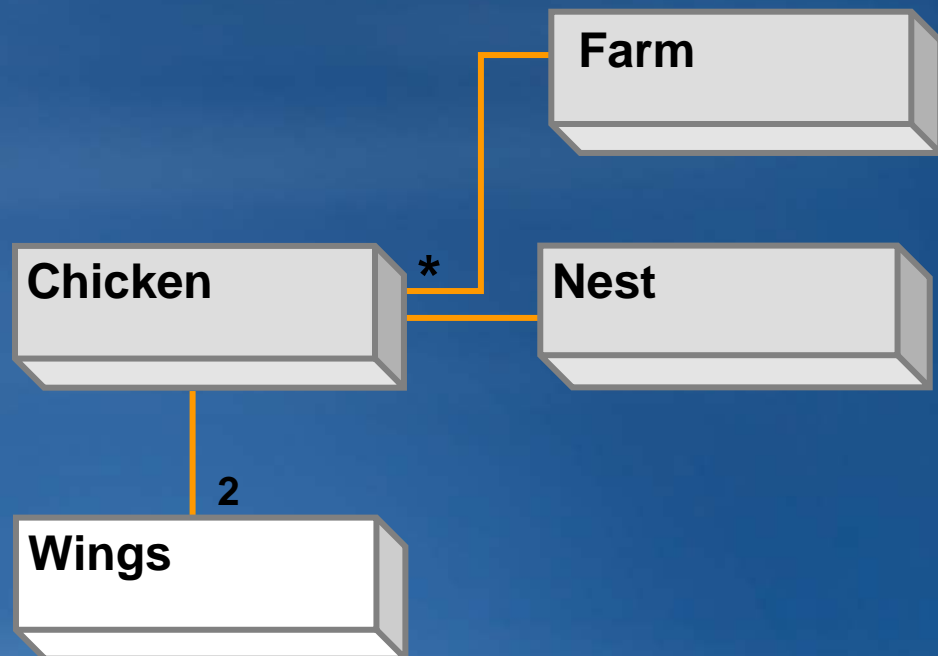
UML symbols

- Multiplicity
- Association



UML symbols

- Multiplicity
- Association



UML symbols

- Creates a
- Multiplicity
- Association



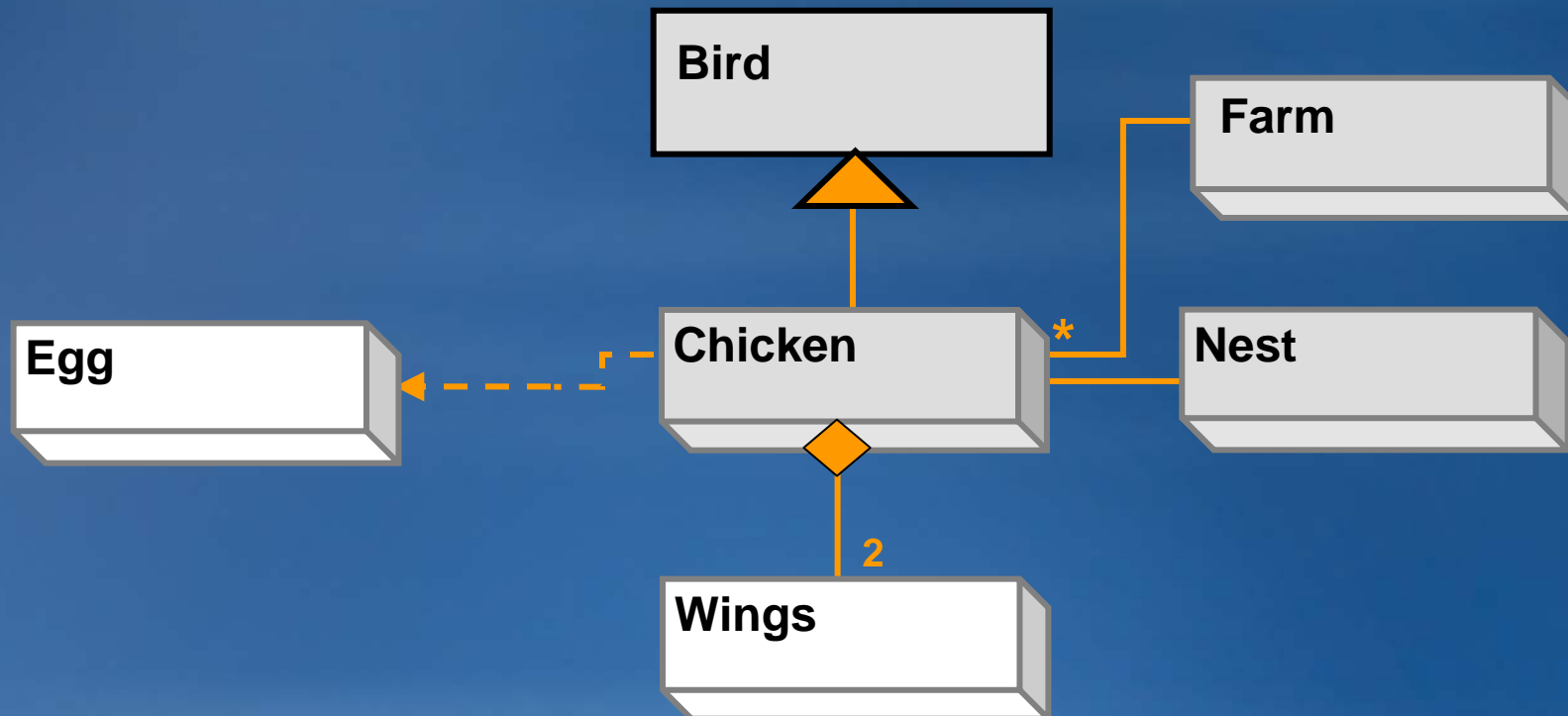
UML symbols

- Is composed of
- Creates a
- Multiplicity
- Association



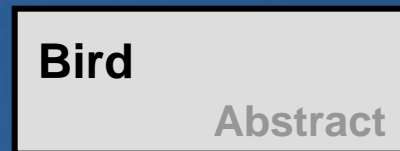
UML symbols

- Is a type of
- Is composed of
- Creates a
- Multiplicity
- Association



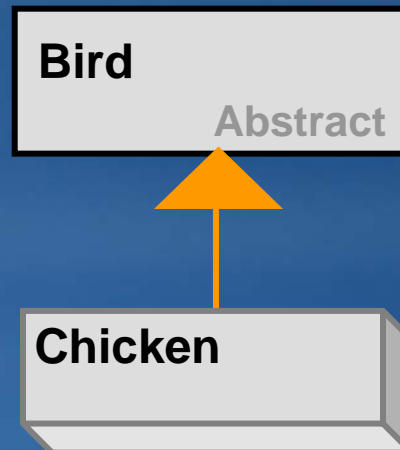
UML class symbols: Abstract class

- 2D and not shaded
- Objects can not be created from it



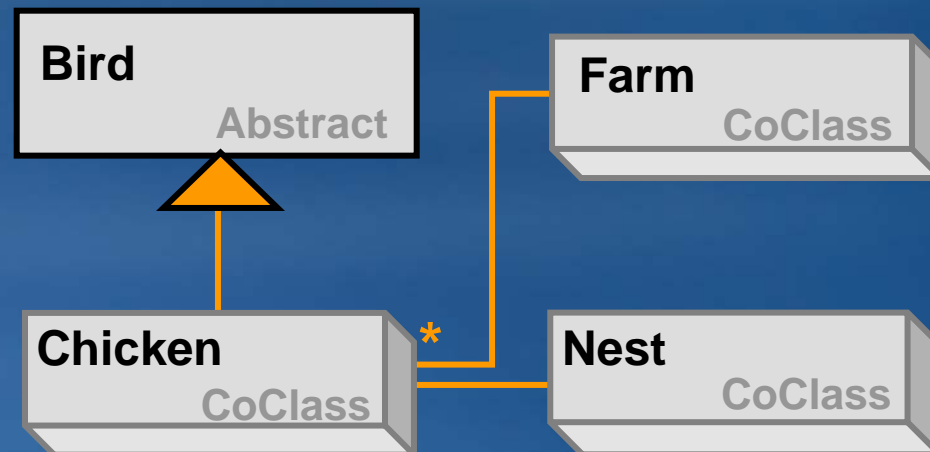
UML class symbols: Abstract class

- 2D and not shaded
- Objects can not be created from it
- Holds properties and methods that subclasses inherit



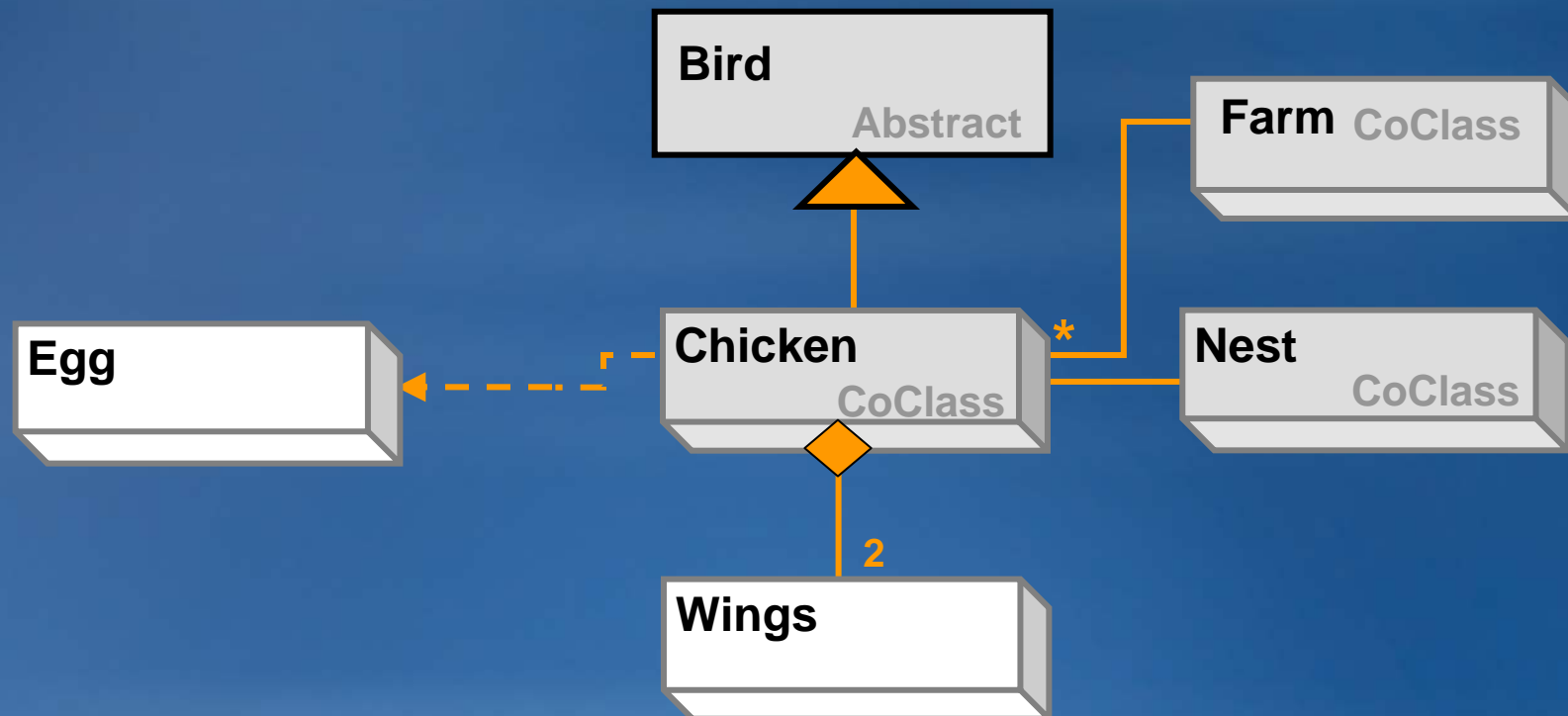
UML class symbols: CoClass

- 3D and shaded
- You create objects out of them
 - Declare a variable
 - Instantiate an object using the New keyword




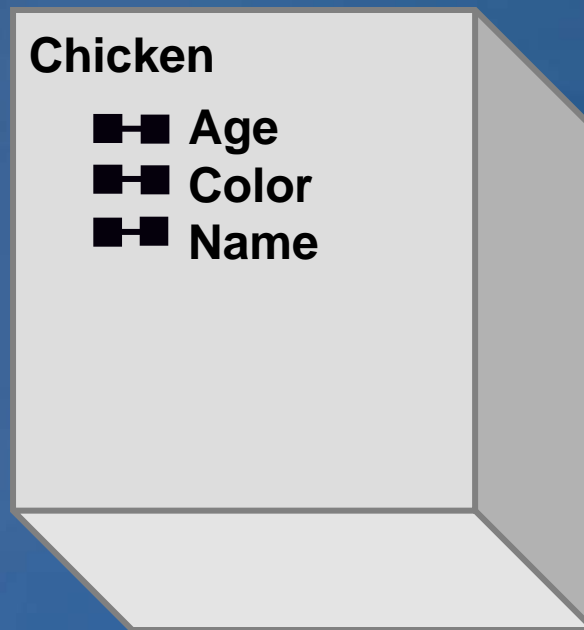
UML class symbols: Class

- Other classes create or return these objects
- You write code with another object to create or get
 - You can't create an egg without a chicken
 - You can't get a wing without an chicken



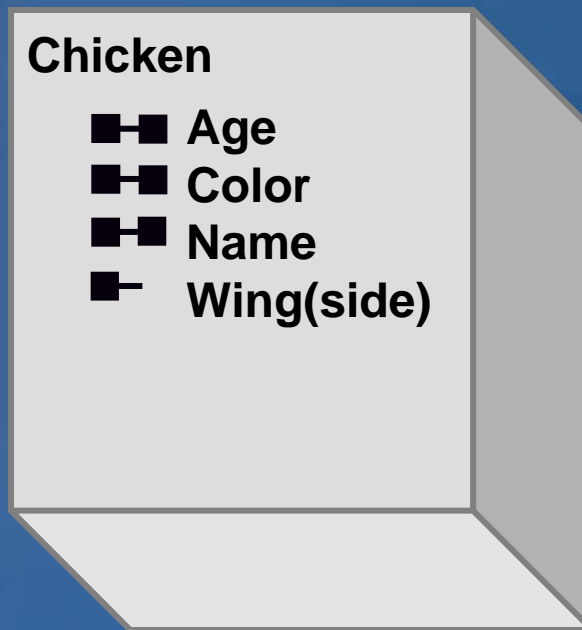
UML property symbols

- Read and write property 
 - These are attributes stored about the object
 - You can either get or set these properties




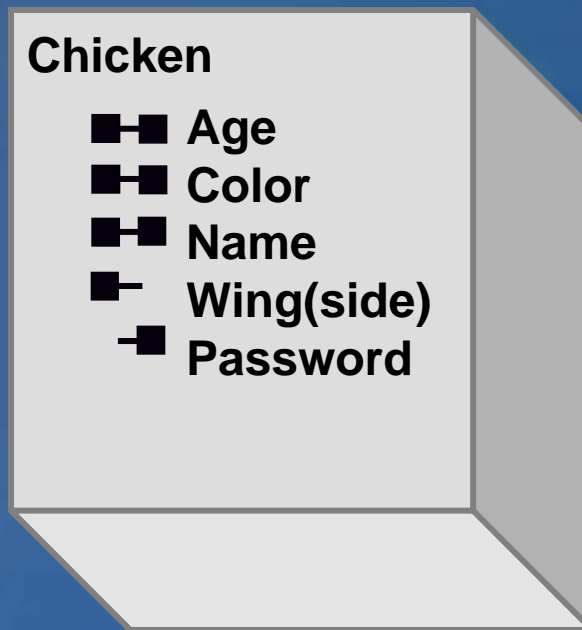
UML property symbols

- Read only property
 - Left half barbell symbol
 - You can get this property's value
 - But you can't change it



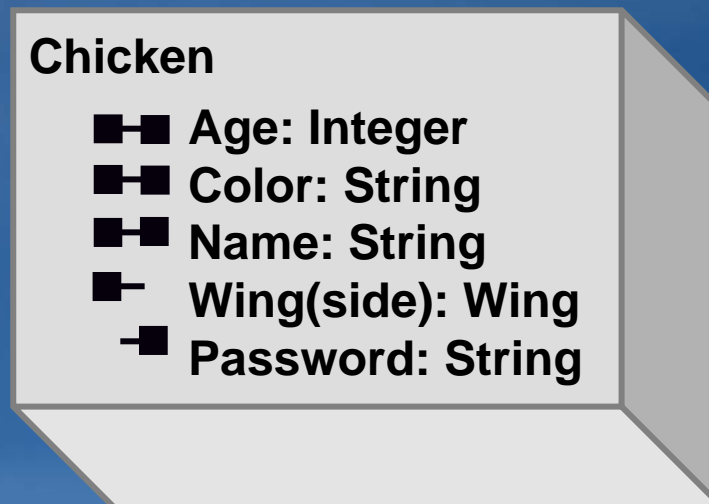
UML property symbols

- Write only property 
 - Right half barbell symbol
 - Usually an edit property or like a password
 - You can change the value, but you can't get it



Property values

- Each property holds a value
- The values are of a certain type: Number, string, date, Boolean, object ...
- The type appears to the right of the property name
 - Property name, a colon, and type
 - Name: String - means that the Name property holds a text string
 - The Wing property holds Wing objects



UML method symbol

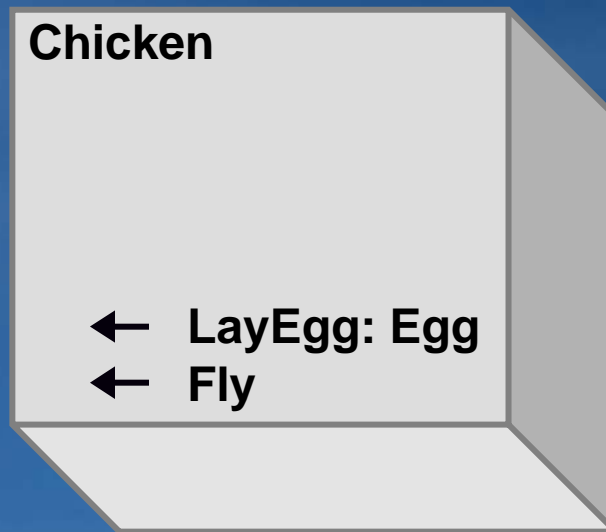
- **Method**

- **Arrow symbol** ←
- **Methods are actions the object can perform**
- **Sometimes called behaviors**



Methods return values

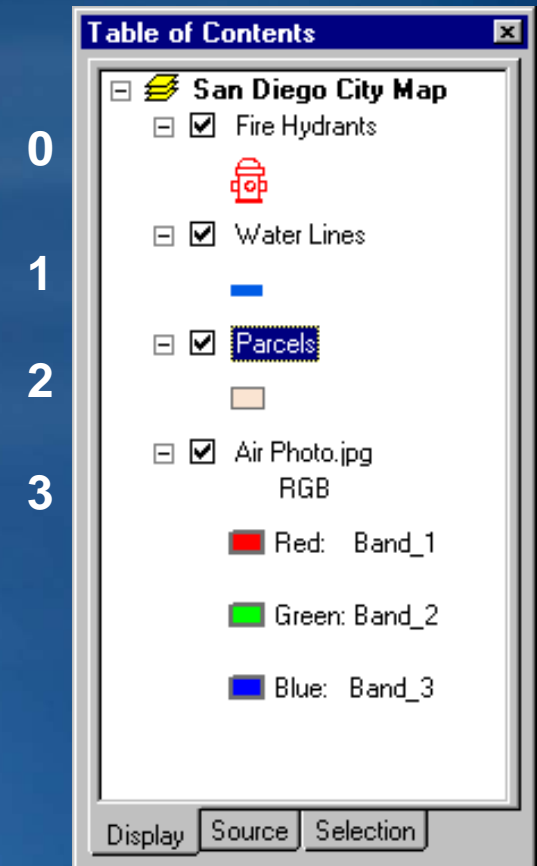
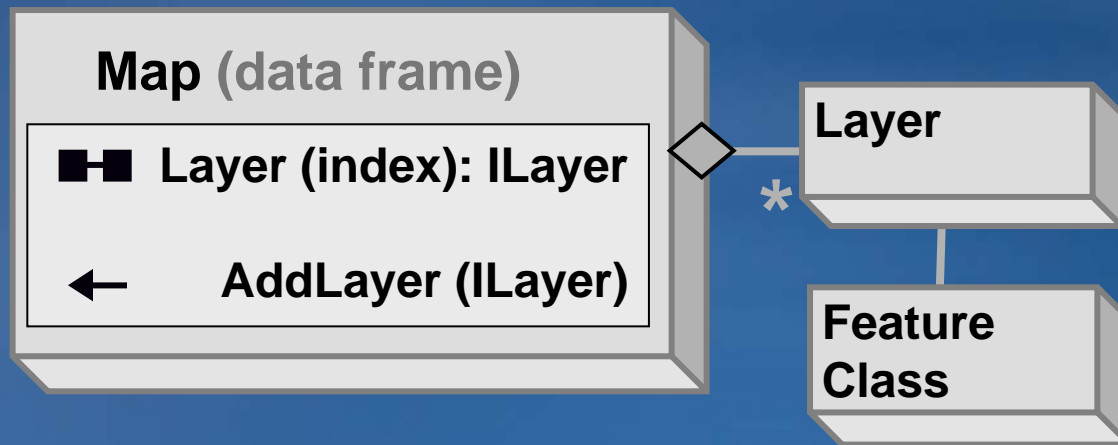
- You write code to run a method
- Some methods return a value, some don't
- The value's type appears to the right of the method
 - Method, colon, and type of its return value
 - LayEgg: Egg - the LayEgg method returns an egg object
 - The Fly method returns nothing



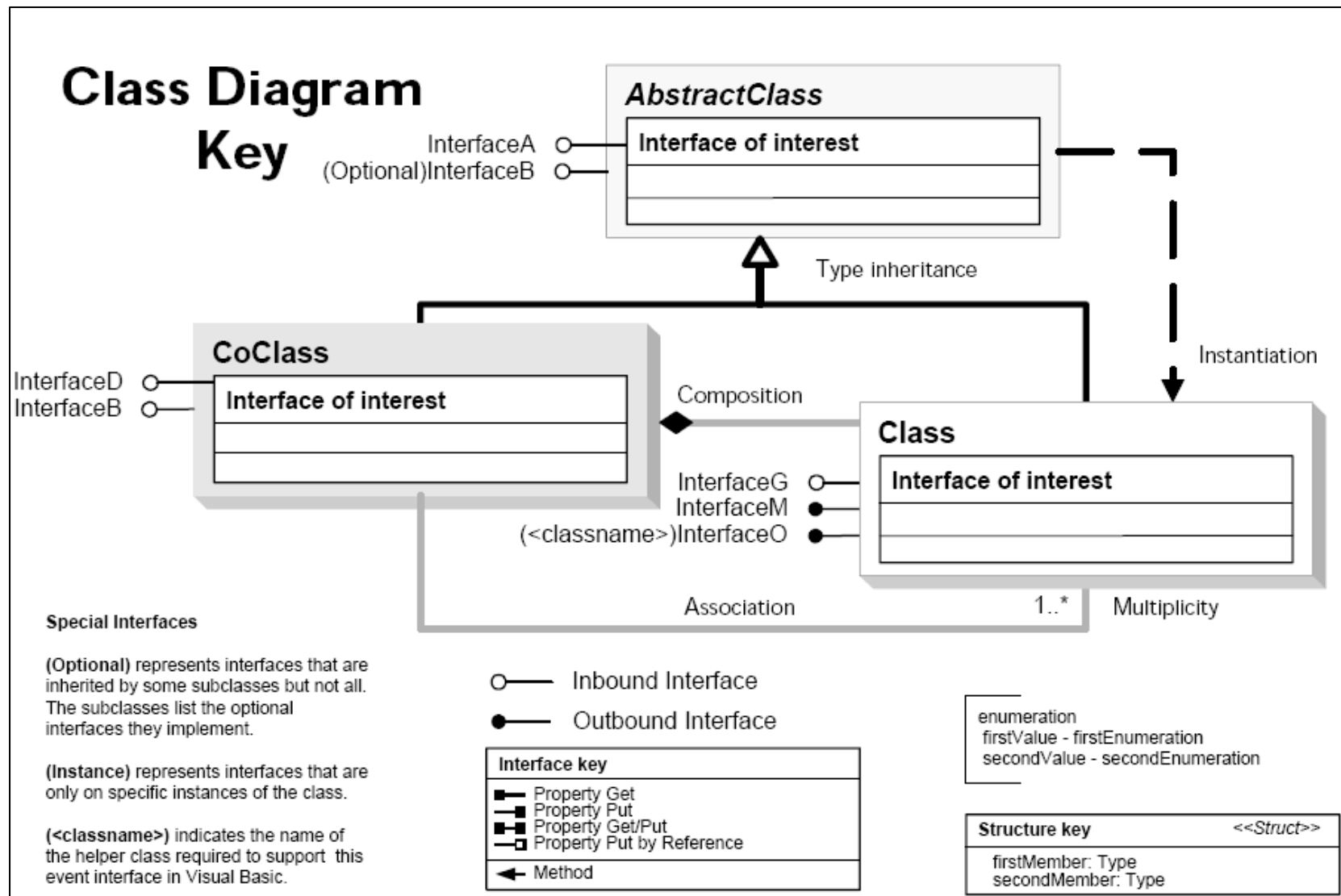
Reading diagrams

- **Classes are rectangles**
- **Classes have properties and methods**
- **Get neighboring or connected objects**

`pLayer = pMap.Layer(0)`

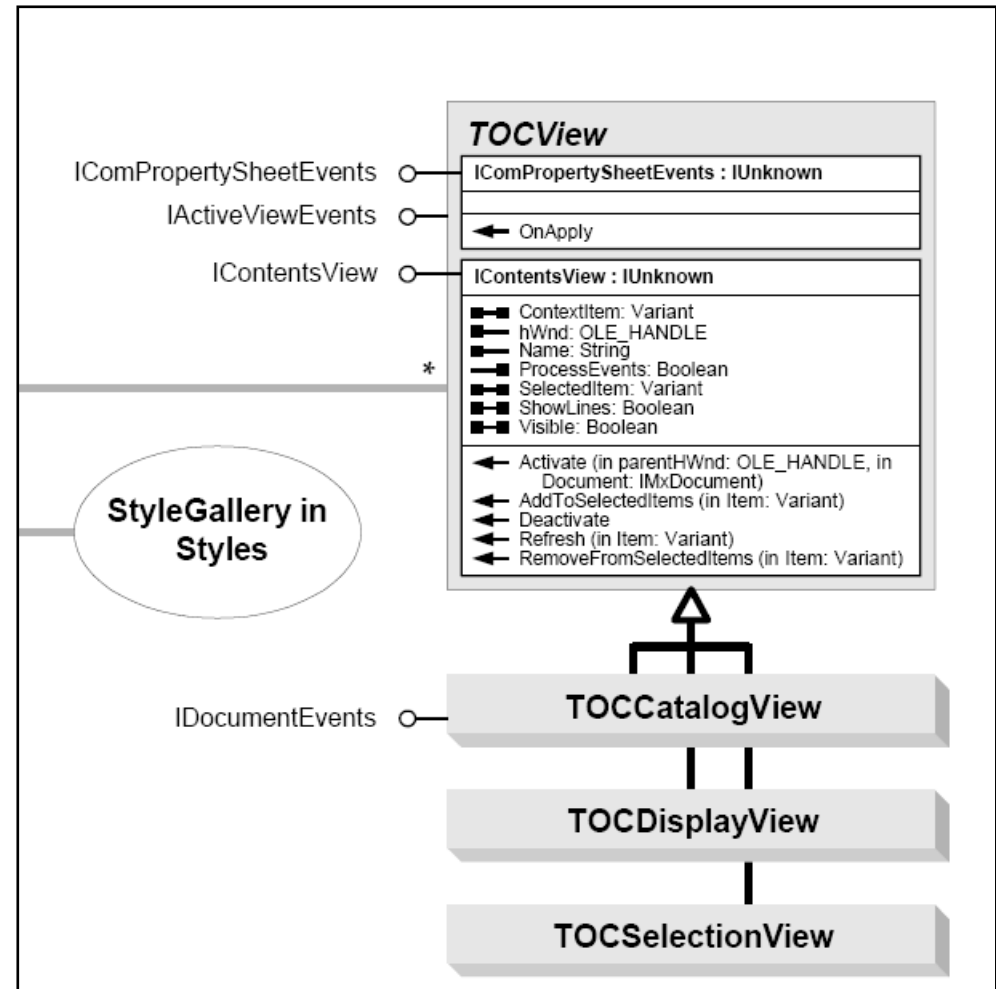


Chapter 11 – Navigating object model diagrams: Reading object model diagrams



Chapter 11 – Navigating object model diagrams: Abstract classes

- **Abstract classes** are symbolized by a **2-D gray box**
- They are **neither instantiable** (using the New keyword) **nor** are they **creatable** (by using requests to other classes)
- They define **general interfaces** for subclasses



Chapter 11 – Navigating object model diagrams: CoClasses

- **CoClasses** are symbolized by a **3-D gray box**

- They are **instantiable**, using the **New** keyword, e.g.:

Dim pMap as IMap

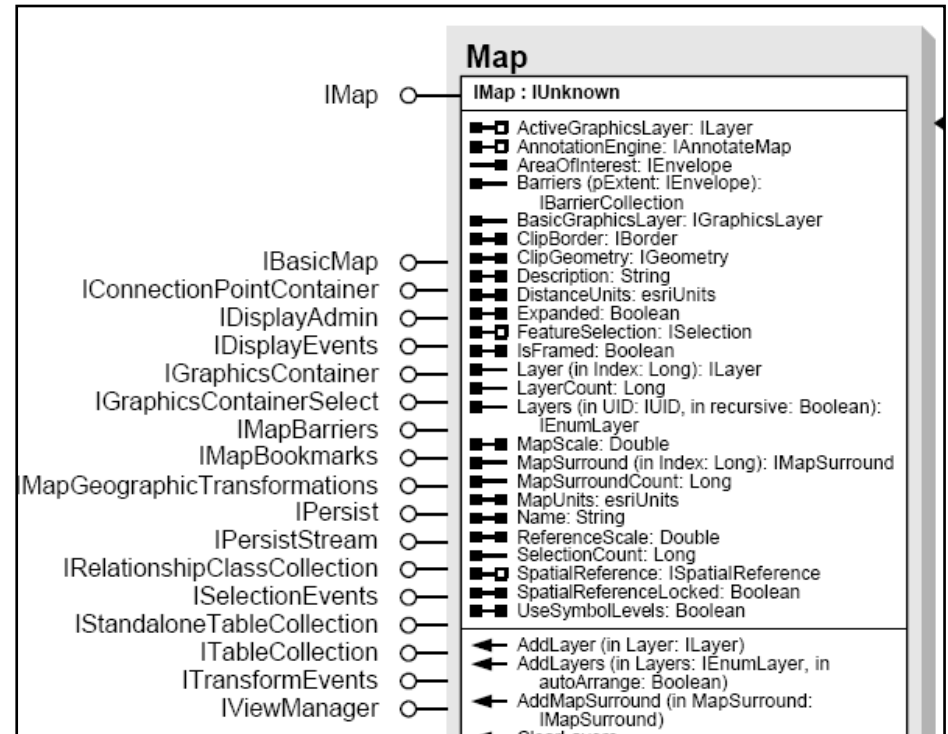
Set pMap = New Map

- They are **creatable**, e.g.:

Dim pMap as IMap

Set pMap =

pMxDocument.FocusMap

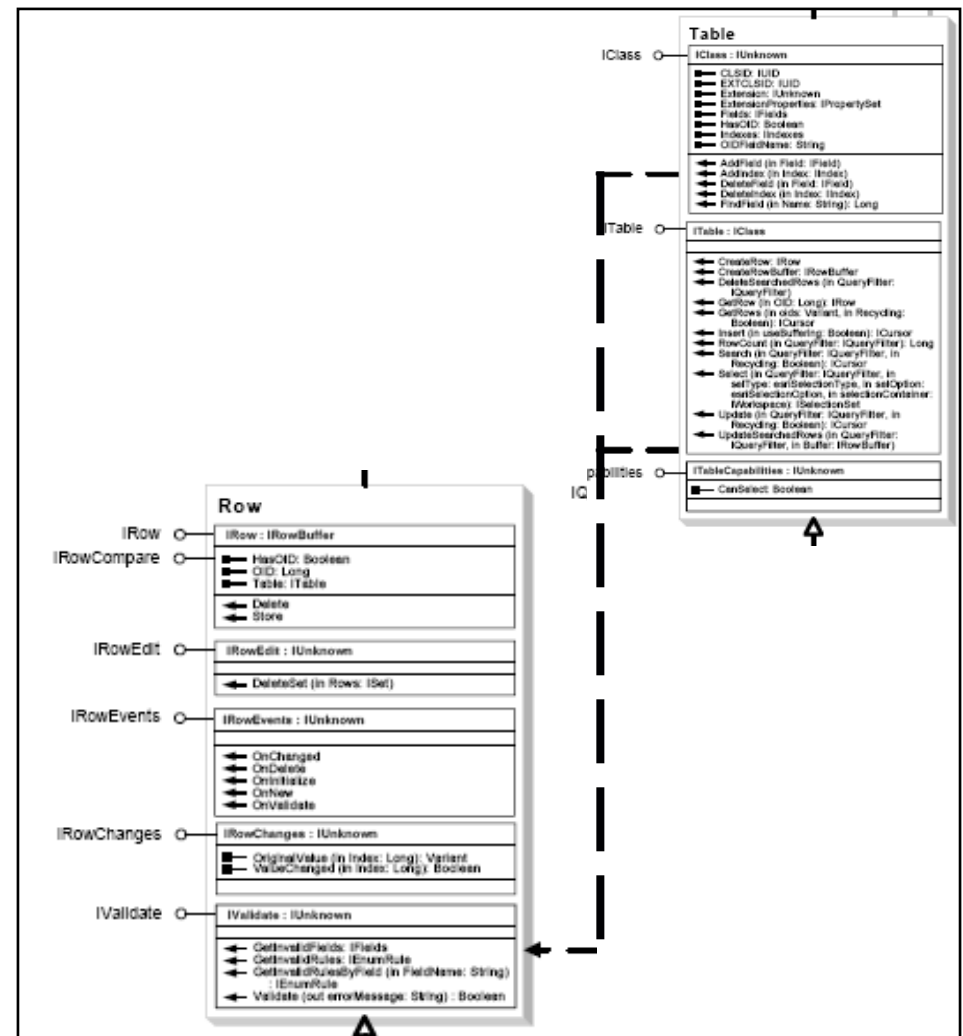


Chapter 11 – Navigating object model diagrams: Classes

- **Classes** are symbolized by a **3-D white box**
- They are **not instantiable** (you cannot make one using the New keyword)
- They are **creatable**, you must **obtain instances from other objects**, e.g.:

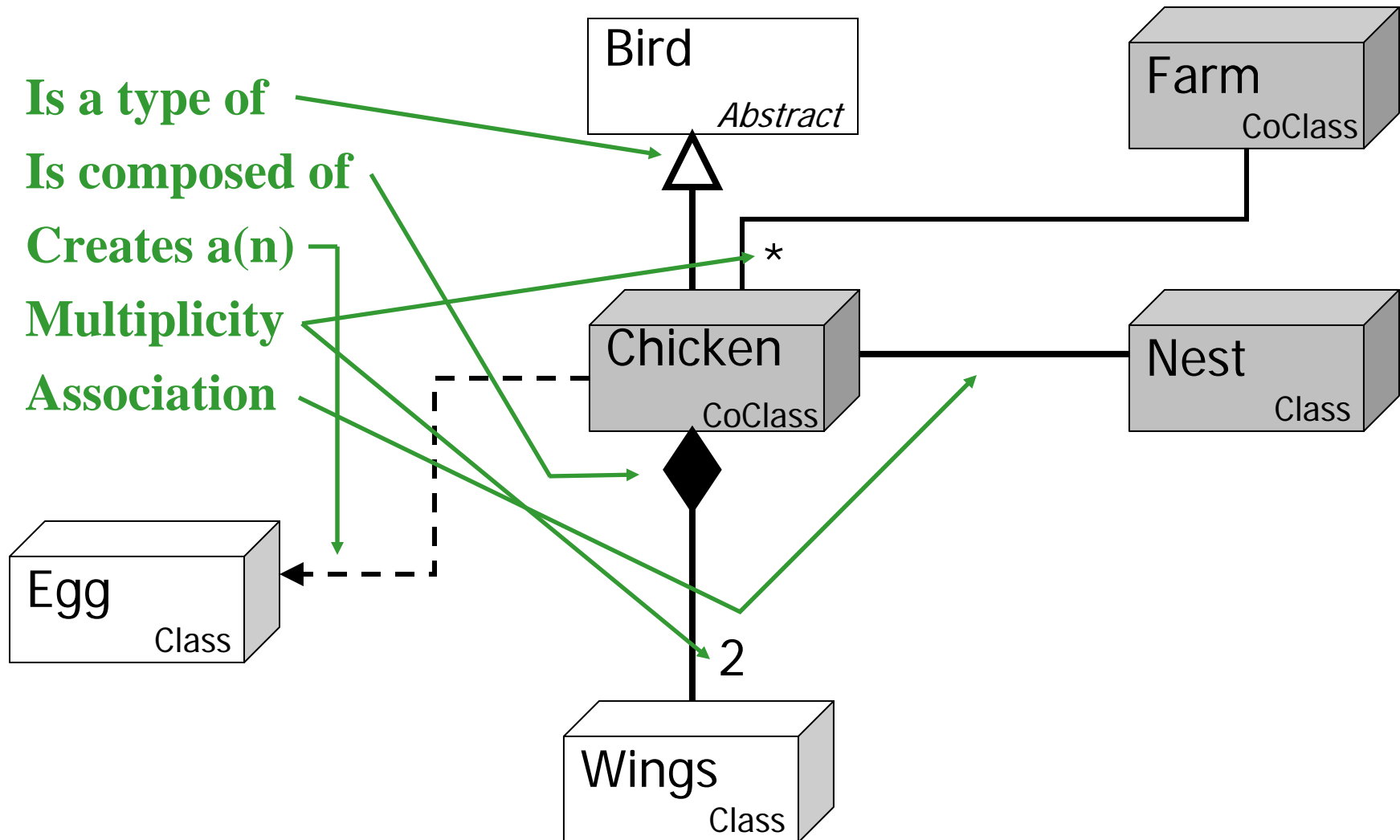
Dim pNewRow as IRow

Set pNewRow =
pTable.CreateRow



Chapter 11 – Navigating object model diagrams: Relationships between classes

- **Is a type of**
- **Is composed of**
- **Creates a(n)**
- **Multiplicity**
- **Association**



Getting layers

- Chapter 11 provides **two exercises** to help you get familiar with **using the ArcGIS object model diagrams**, navigating from object to object to get **from the ones you have to the ones you need**
- The first exercise is about **getting layers**, a common task if you are working with map documents
- Recall from Chapter 10 that we can **always count on having the Application object and an MxDocument object** to start with when you work in ArcMap
- In the exercise, you'll start with the MxDocument and **navigate from that to the layers**, finding the path in the object model diagram, and writing code based on it

Creating and assigning colors

- The second Chapter 11 exercise gives you some practice in changing the symbolization of a map by **modifying the colors of map elements in a layout**
- Once again, you will **start with objects that you have** from the outset, and **use object model diagrams to find a way to get the objects that you want**, so you can change their properties and use their methods
- While these exercises will give you exposure to some commonly useful objects, the **key** here is the **skill** you are developing; the **ability to use an object model diagram to find the relationships you need** so you can use them in your code to do what you need to do!

Chapter 12 – Making tools

- Reporting coordinates
- Drawing graphics
- Using TypeOf statements

Chapter 12 – Making tools

- On multiple occasions earlier in the course, it has been mentioned that **tools** in ArcGIS **are different from buttons**
- This is obvious even from the **user's point of view**: **Clicking on a button** causes ArcGIS to **do something immediately**, whereas **clicking on a tool changes the appearance of the cursor** ... and then the user then use the mouse to control the cursor to use the tool to do something
- As a budding ArcGIS programmer, you probably can guess that **developing the code for a tool** is going to be **more complicated** than it is for a button

Chapter 12 – Making tools

- The **key procedure** for a **button** is the code associated with its **click event**
- But for a **tool**, which the user can interact with in a number of ways, there are **many more events to code**
- And beyond the number of events, developing a tool requires you to have a **broader understanding of a variety of objects** (maps, layers, geometry like points that specify the position of the cursor)
 - Hopefully you are **becoming familiar with these many objects** and even if you are not ...
 - Hopefully you now **know how to use the UML object model diagrams** to find out the things you need to know

Reporting coordinates

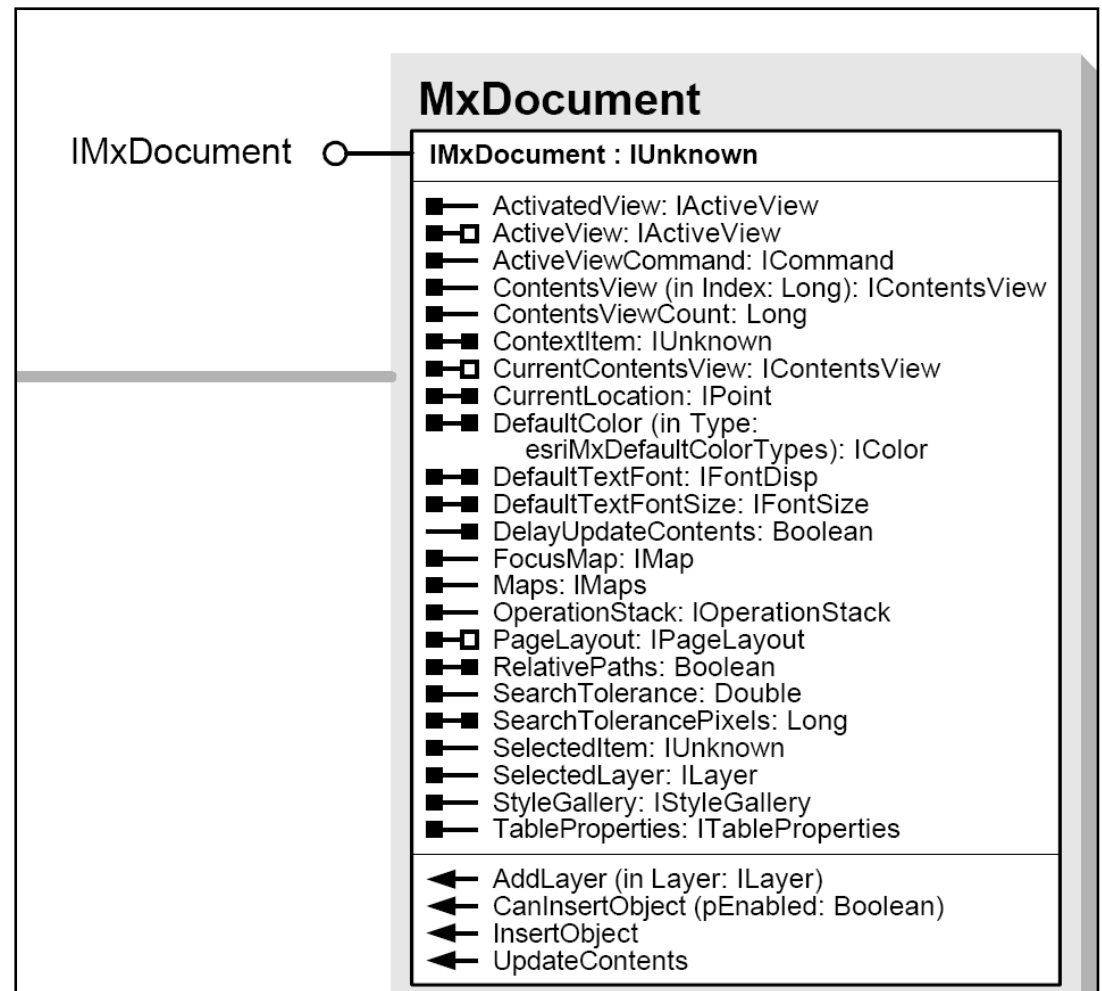
- Tools have a **MouseMove event procedure** that runs whenever the user moves the cursor with the tool selected
- This procedure takes **four arguments**, that the user specifies by using the mouse:
 - button As Long
 - shift as Long
 - x As Long
 - y As Long
- Each of these integers is a **value that represents some part of the mouse state**: Button and shift reflect whether the button or shift key is depressed, x and y report the position in pixels of the mouse pointer

Reporting coordinates

- With the button and shift variables, **If Then statements** can be used to **create appropriate code** for the various permutations
- Further events like **MouseDown and MouseUp** respond to pressing or releasing the mouse button
- Note that the **x, y** reported here are **pixel positions** in the map display which (of course) are **not in geographic coordinates** ... but fortunately, we can **navigate through the object model** to find the appropriate objects, interfaces and properties to get the position of the map pointer in geographic coordinates

Reporting coordinates

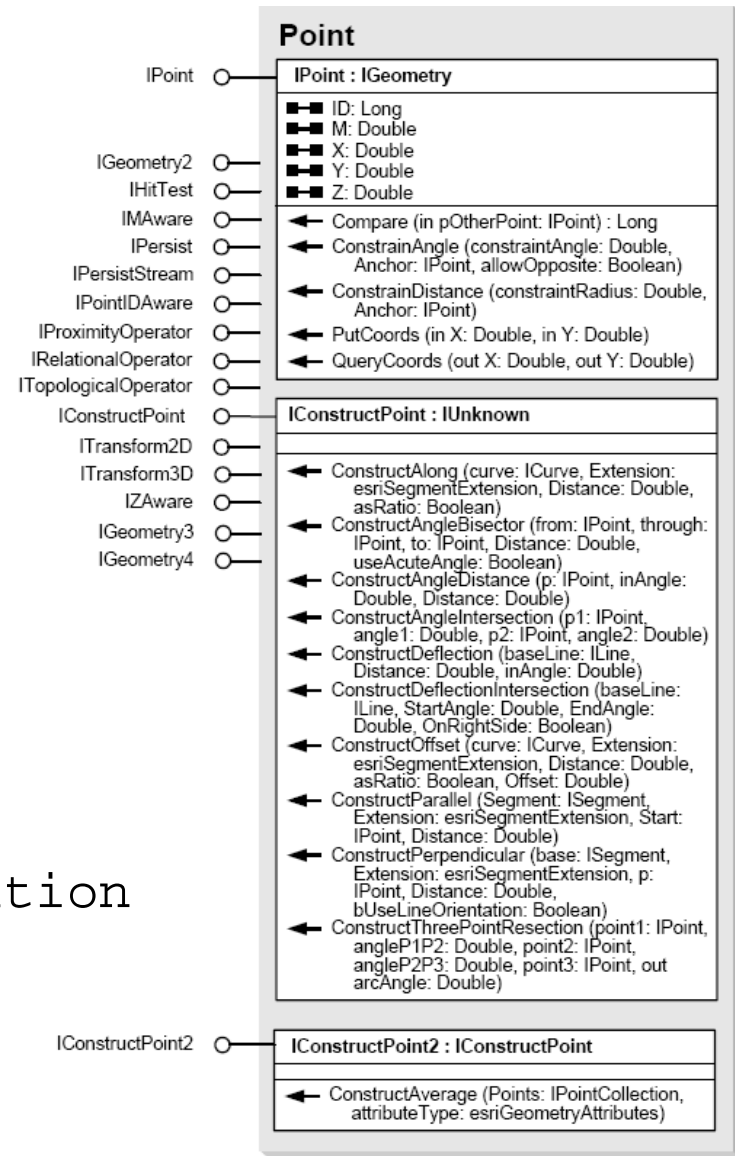
- The **MxDocument** class has a property **CurrentLocation** on its **IMxDocument** interface
- That property will provide the **position** of the mouse pointer in **geographic coordinates** using the **IPoint** interface



Reporting coordinates

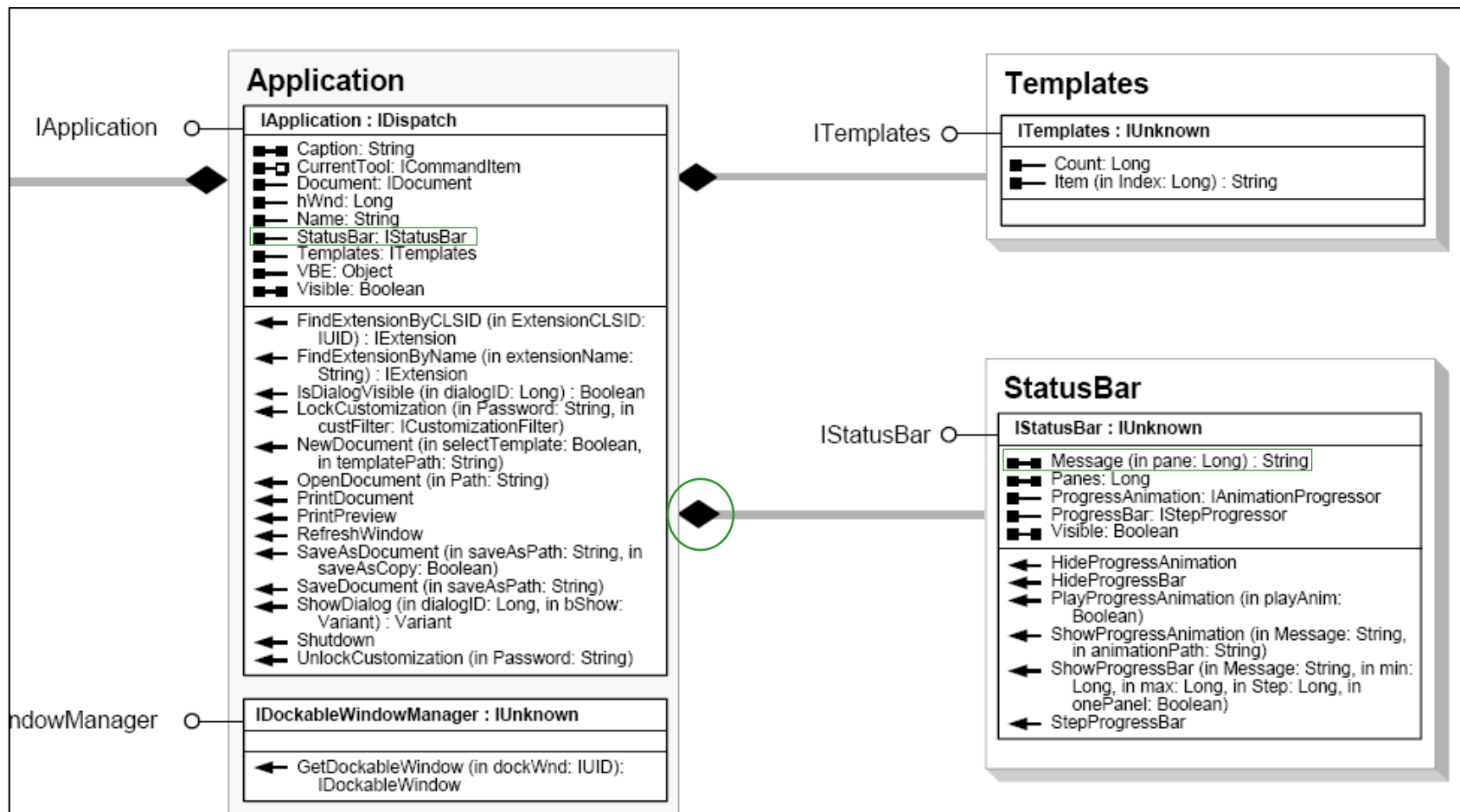
- We can search the developer help and find that the **IPoint interface** is available on the **Point coclass**, and can thus obtain the geographic location of the mouse pointer with:

```
Dim pMxDoc As IMxDocument
Set pMxDoc = ThisDocument
Dim pPoint As IPoint
Set pPoint = pMxDoc.CurrentLocation
```



Reporting coordinates

- If we want to **report the coordinates to the user**, we can use the **status bar** at the bottom of the ArcMap window:

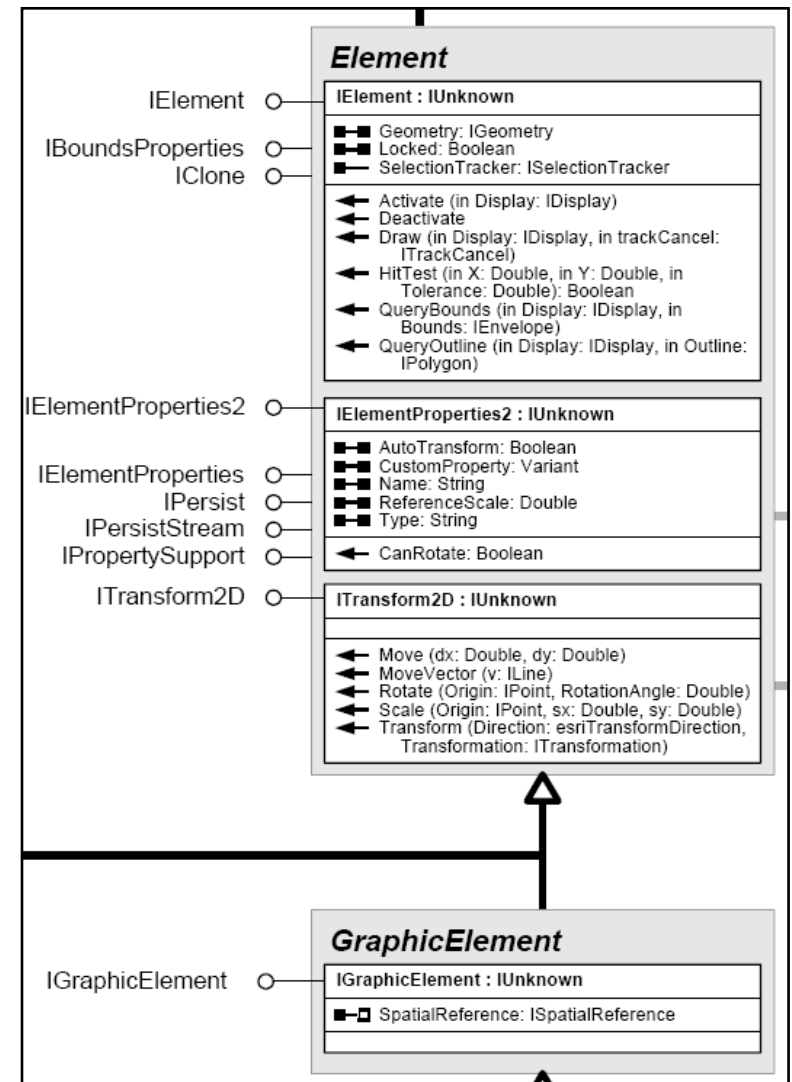


Drawing graphics

- If you have produced a few maps in ArcGIS, you probably know that in a map it is often useful to supplement the geographic data derived from layers with a few **graphics**
- In the second exercise for Chapter 12, you will write some **code to place graphics on a map** where the user clicks
- It will be useful to us to know a little about the **classes associated with graphics** so you can understand how to go about developing this capability in ArcGIS VBA code
 - You will notice an emerging theme here: **Navigating the object model to find what we need** is the key to being able to build what we want to build

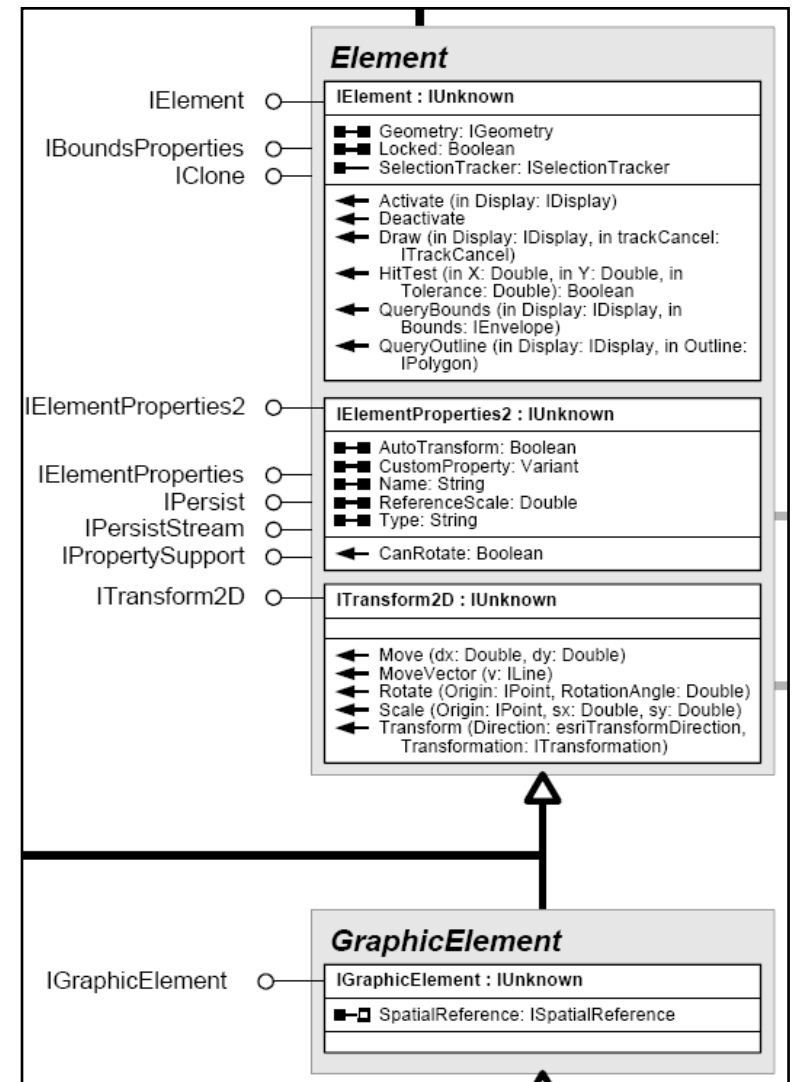
Drawing graphics

- Graphics belong to an abstract class called **Element**
- Element, has **two abstract subclasses** (FrameElement & GraphicElement)
- We are interested in **GraphicElement**, which in turn has coclasses under it named **MarkerElement**, **LineElement**, **PolygonElement**, and **TextElement**



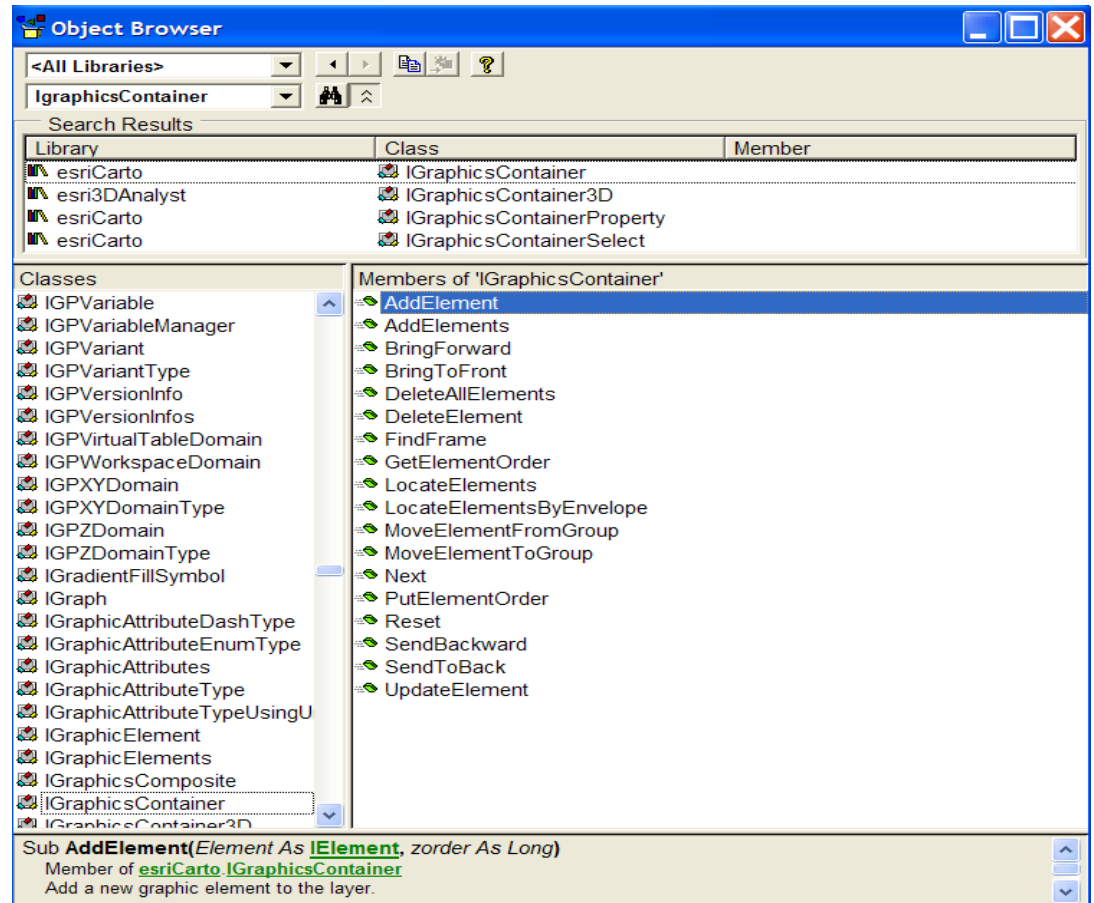
Drawing graphics

- Note that the abstract element class has a **Geometry** property
- This means that **geometry objects** (like points, lines, and polygons) are **associated with elements** (MarkerElements, LineElements, and PolygonElements respectively)
- We thus can create appropriate **geometry objects** (like points) and use them to **position marker elements** on a map



Drawing graphics

- Adding a graphic to a Map is done through the Map's **IGraphicsContainer** interface
- Once added, **refreshing the Map** causes it to redraw itself, including the graphics associated with it (see the text for details)



A Shortcut for QueryInterface

- Up until this point, when we have wanted to **switch interfaces** (also known as the **QueryInterface** operation), we have had to use **two extra lines of code**, e.g.:

```
Dim pMap as IMap  
Set pMap = pMxDoc.FocusMap  
Dim pGraphics As IGraphicsContainer  
Set pGraphics = pMap
```

} Here we have an MxDocument, and we get the IMap interface from FocusMap

} QueryInterface to switch from IMap to IGraphicsContainer

- It is possible to do this in **two lines instead**, by declaring **the interface we need** from the outset, and allowing VBA to make the interface switch for us:

```
Dim pGraphics As IGraphicsContainer  
Set pGraphics = pMxDoc.FocusMap
```

Using TypeOf statements

- Once you have developed your tool for drawing graphics at rescue sites in the Map View in Exercise 12B, we have **a problem**:
 - This tool **would not work properly in the Layout View**, which does not operate in geographic units
- We need a way to **distinguish between Map and Layout Views** to turn the tool on and off appropriately, and we explore this in Exercise 12C, **using TypeOf statements**
- In this example, and in a diverse set of other situations where having an **object of the wrong type** would **break our code** (and return a type mismatch error), we can use **TypeOf** (which returns TRUE or FALSE) to check if an object is the required type

Next Topic:

Using existing commands
and adding layers