Object Modeling

BRIAN LAMARCHE

COMPUTER SCIENCE 323 – SOFTWARE DESIGN

Objects

An object is the most basic aggregation of attributes and methods that describe an abstract or physical concept.

- E.g. Target, Camera,

Terminology

- **Attributes:** Class variables, etc.
- **Methods:** Functions

Objects are intended to organize data and **encapsulate** it (protect) from external entities.

Objects are types.

All objects in *.net* derive from the type *object* (e.g. int, Target, double, etc.)

- *Although we have value (copied as an argument) vs. reference types. Deferring discussion here*

Object Scoping

We encapsulate attributes and methods using scoping keywords:

- Public – any one can see the attribute or method
- Private – only visible from within
- Protected – only derived object can access or see the attribute or method
- virtual – for derived types to **override** functionality

In C#, we also have

- internal, but I will defer discussion of this to you

Object Methods

We describe an object's method by it's signature:

- Scope
- Return Type
- Name
- Parameters
  - Type
  - Variable

public void ShootMissile();
public void ShootMissile(int x, int y);

*The above is called **overloading**.*

public override void FireZeMissiles();

*The above is called **overriding** when base type has a virtual method*

Object Methods

We will say that an object has an **interface**, meaning the collection of public or protected method signatures.

Basically, how an object interacts with other objects.

I want to make an distinction now, that an interface is a commonly used term in programming. For this lecture, we may use it interchangeably. If you are confused raise your hand.

Objects are both abstract and real

Objects can be real entities: Target

A target is a physical thing that we can see feel and touch.

But a target has a shape. And the shape is an abstract mathematical concept.

However, our software may need to operate on various shapes

Shape.Color(Red);

Shape.Draw();

In advanced data structures we might describe a target and shape with an object hierarchy.

Example: Shapes

Circles and Squares are shapes

They may share common behavior

- Draw

They may share common data

- Polygon data

We don't care about draw implementation, that is for each shape. We just care about the shapes.

Example: File Readers

File readers are another example of an object abstraction.

- XMLReader, INIReader

Both of these objects are tangible, they serve a purpose, but they do one conceptual thing:

Read Files

These types of behaviors is captured through object abstraction

Is-A

First consider the relationship between a Friend Target and a Foe Target

FriendTarget.Draw()

FoeTarget.Draw()

A Friend Target **is a** target

A Foe Target **is a** target

Both are of type target. We use the ***is-a*** rule to help us establish object abstractions.

Object Abstraction

You should already know about abstract objects, virtual methods, etc. from advanced data structures.

Abstract classes

- Offer a basic structure for what an objects interface (collection of methods) should look like.
- Can provide implementation
- Retain only information specific to its abstraction

Shape

Color

IsFilled

Points – protected data

Draw (virtual) – does not draw anything

Terminology

Extends
- When a class sub-classes a super class.

Superclass
- A base class that defines a set of functionality and attributes

Subclass
- Class that derives from a super class, extending functionality and attributes

Polymorphism
- the ability for an object or method to take multiple forms
  - *POLY – multiple*
  - *Morphic – shape*

Inheritance
- The process of which an object uses re-uses existing code (i.e. objects) by sub-classing.

Terminology

Purely abstract

- **Base class that provides only the method signatures but no implementation (C++).** *In C# we have interfaces.*

Abstract Class

- Base class

Concrete Class

- Class that derives from the abstract class and provides full implementation

Language Caveats

In Java and C#, objects can only derive from one class.

public class Labrador : Animal – OK

public class Labrador : Animal, Pet – Not OK

But this does not mean that objects cannot have multiple inheritance hierarchies:

Dog derives from Pet derives from Animal

Danger of Abstraction

The biggest danger with using object hierarchies is over-exploiting a hierarchy

….Labrador derives from Pet derives from Dog derives from Animal derives from LifeForm derives from …..

Be careful how far you go in your tree

Ask:

is X really ***a*** Y

In C# we have a keyword *sealed* that stops objects from being sub-classed.

Example: C# Code of Animal

See code examples: https://github.com/brianlamarche/Cpts323

Example: C# Code of Animal

```
public  sealed class Goat: Animal
{
        // Implementation
}
```

Using the keyword sealed we prevent other classes from building on this class hierarchy.

MSDN: http://msdn.microsoft.com/en-us/library/88c54tsw(v=vs.71).aspx

Object Modeling

We use the UML Class diagram to show the relationships between objects.

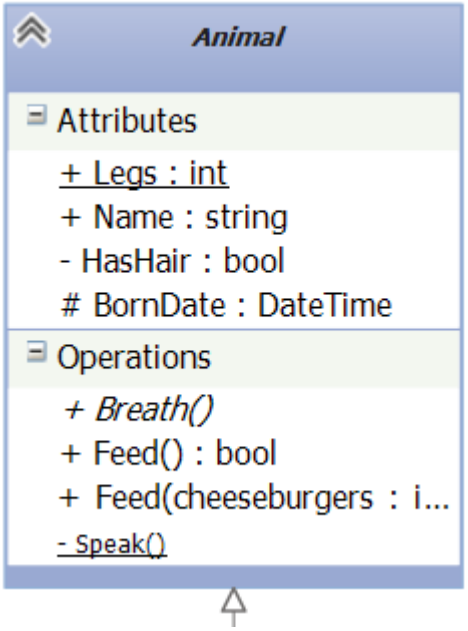
This is especially useful to show the relationship between derived and base class objects.

UML Basics

We want to focus on these things:

- Scoping
- Modifiers (e.g. static)
- Method names
- Parameter names and types
- Class names
- Return types
- Associations
  - Relationship to other objects

UML Basics



Scoping and Modifiers

- \+ is public
- \- is private
- # is protected
- Underline means static
- *Italic* means abstract

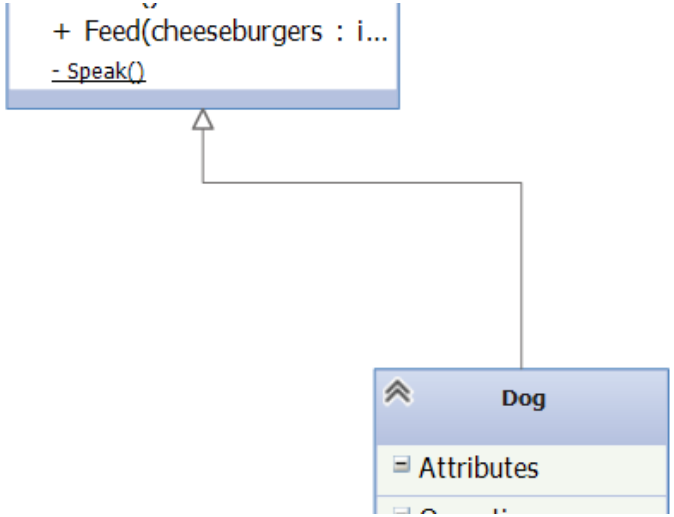
Attributes:

- *<scoping> Name : type*

Methods:

- <scoping> MethodName (parameters : type) : return type

Example: Abstract Class

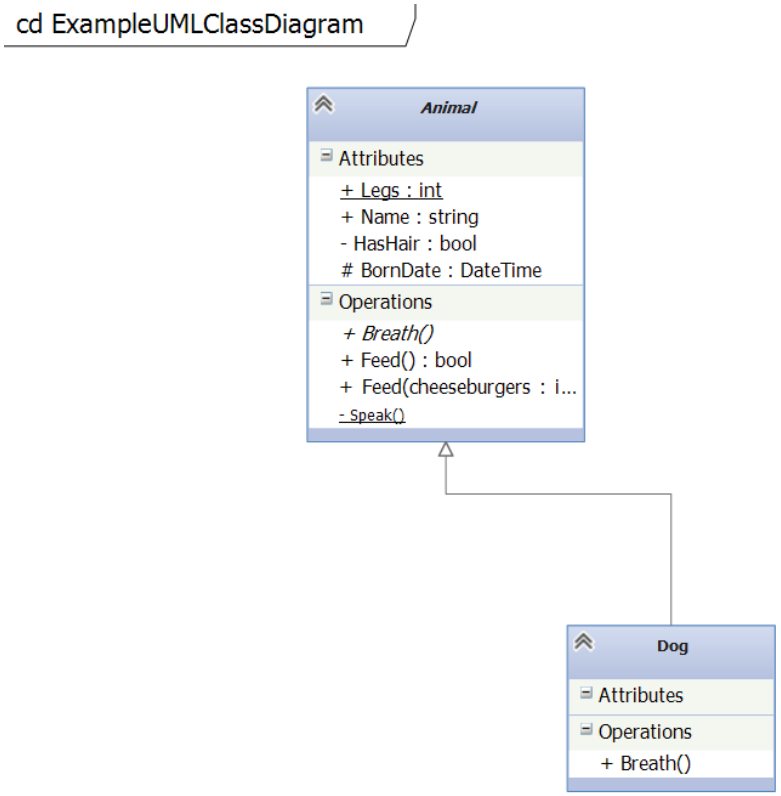


Dog inherits from an Animal

Triangle points to the superclass

Solid line shows relationship

Example: Abstract Class

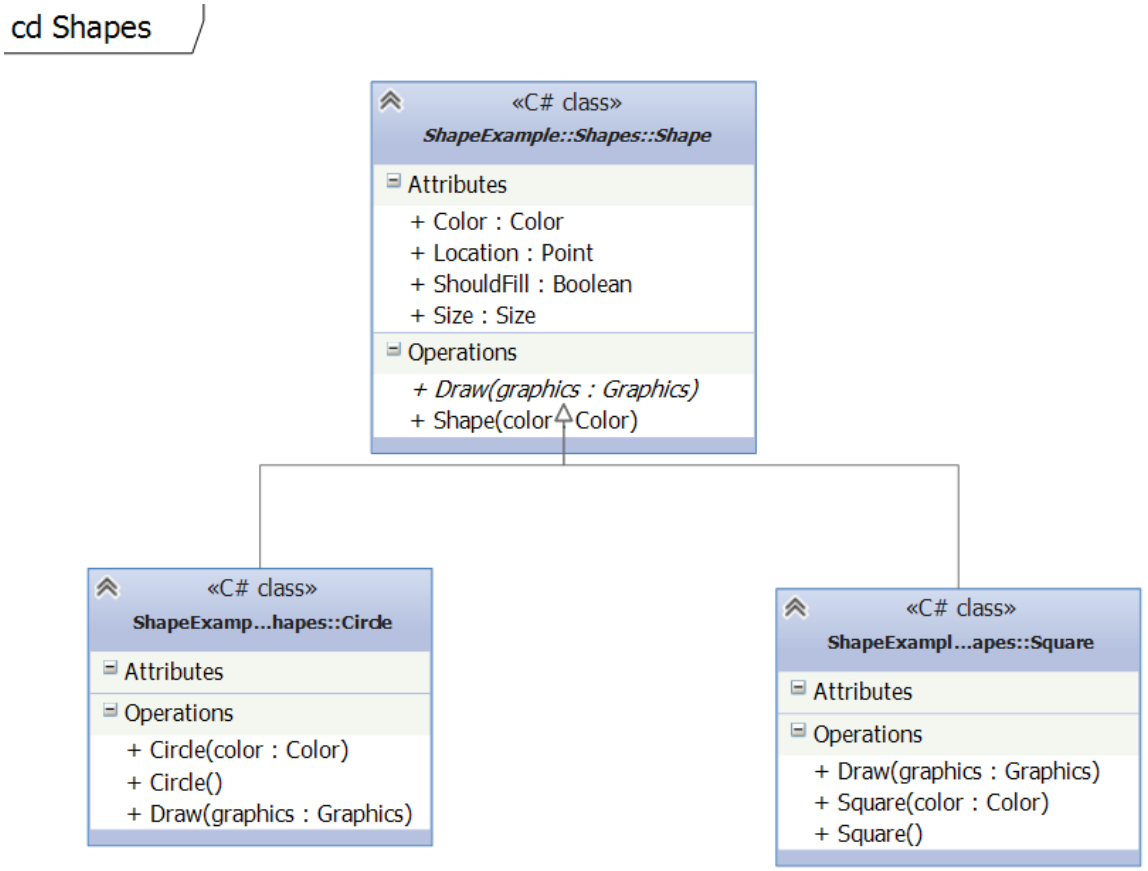


Dog inherits from an Animal

Triangle points to the superclass

Solid line shows relationship

Example: Shape Class



UML Class Diagrams

UML Class diagrams are very useful

They help describe the relationships between many objects

A class can be first modeled, then used to generate code via VS

Other tools do this as well

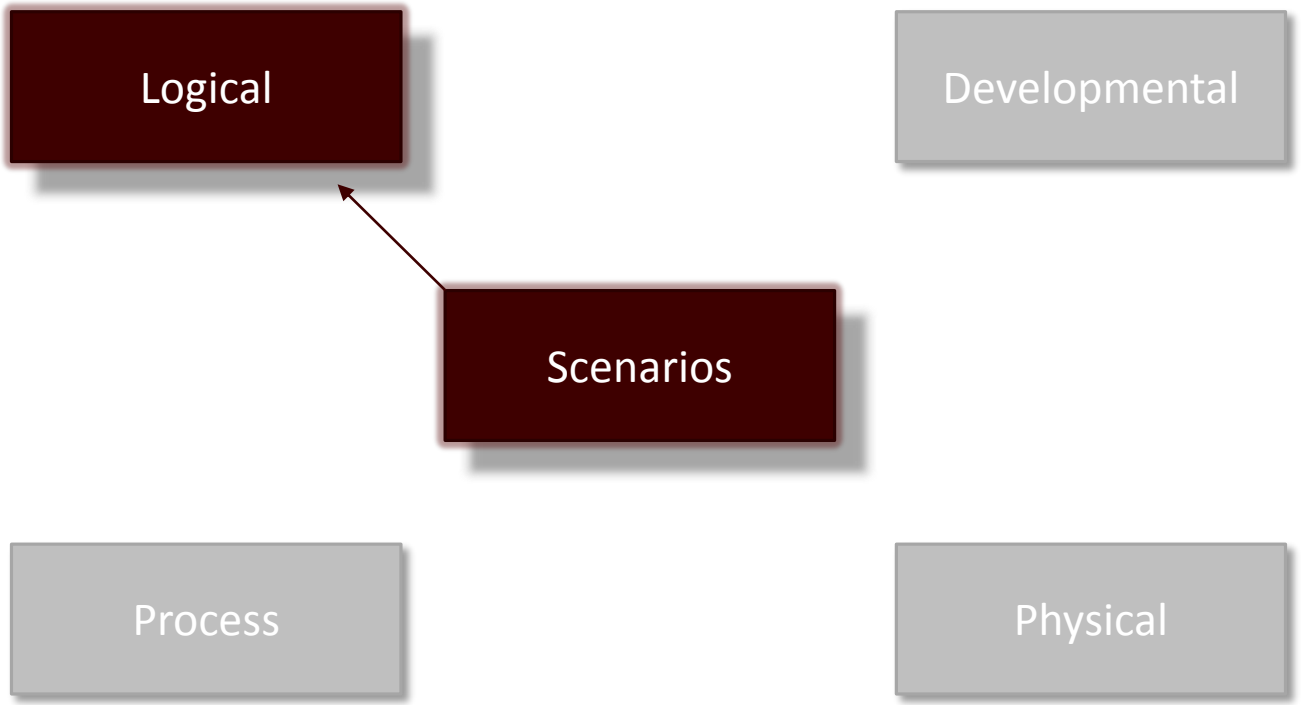
You can generate UML Class diagrams from your code also

Apply this to a file reader

Now that you know how to create an abstract class and how to model it, apply it to an XML file reader and INI file reader.

Why would you prefer to reference a base class instead of a concrete class?

Tying back to 4+1



Tying back to 4+1