

Short Inheritance Questions

- o If class Tornado inherits from class Storm, mark the following statements as true or false:
 - o There are more objects that conform to the type Tornado than there are that conform to the type Storm.
 - o Tornado instances probably have more fields (attributes) than Storm instances, and definitely not fewer.
 - o All methods declared in Storm can be called on instances of Tornado.
 - o All methods declared in Storm will perform the same way whether they are invoked on instances of Storm or of Tornado.

Abstract Classes

- o An abstract class is a class from which no instances can be generated.
- o Abstract classes can contain abstract methods: methods without implementation

There are Silly Geese

- o Please see the Inheritance file at:
www.cs.rit.edu/~cs2/schedule.htm
- o Write the silliest Goose subclass you can think of. Keep it simple!

Why won't this compile?

```
using System;
public class TalkingGoose : Goose {

    public TalkingGoose ( string name ) :
        base(name, true) {

    }

    // have this goose speak its name
    public override void makeNoise ( ) {
        Console.WriteLine( name );
    }
} // TalkingGoose
```

Polymorphism

- o In its simplest form, polymorphism allows a variable of type X to refer to any object that is an instance of X or an instance of a subclass of X.

Polymorphism (cont'd)

- o All classes are descendants of System.Object. So, a variable of type System.Object can refer to an instance of any class.
- o That's why Console.WriteLine can take any argument and print it: it just uses the object's ToString() method to get the format for printing the object
- o You should almost always override the ToString() method in a class

A Polymorphic Method Call

```
class TestGoose3 {  
    public static void Main ( string[] args ) {  
        AbstractGoose aGoose;  
        UncleGoose myGoose = new UncleGoose( "Snookums");  
        System.Object anotherGoose;  
        WolfGoose wolfie = new WolfGoose( "Tigger" );  
        Random rand = new Random( 42);  
  
        if ( rand.NextDouble() < 0.5 ) {  
            aGoose = myGoose;  
        } else {  
            aGoose = wolfie;  
        }  
    }  
}
```

(cont'd)

```
// The next line is a polymorphic method call:  
// Which method gets called depends on the type of  
// object that aGoose refers to. This will be  
// determined during run-time.  
aGoose.makeNoise();  
anotherGoose = aGoose;  
  
// This line won't compile: why?  
anotherGoose.makeNoise();  
}  
} // TestGoose3
```

Yet more casting

- o This won't compile:

```
Goose aGoose = new WildCat( "Tigger" );  
WolfGoose myGoose = aGoose;
```

- o This will compile, but will throw an exception/error:

```
AbstractGoose aGoose = new UncleGoose( "Sam" );  
WolfGoose myGoose = (WolfGoose) aGoose;
```

Using Inheritance

- o Use inheritance to implement an "is-a" relationship

- o If A is a B, then make A a subclass of B
- o GrannyGoose is a type of Goose, so it is a subclass of Goose

- o Do not use inheritance to implement a "has-a" relationship

- o For example, a bird has feathers, a beak, toys, etc. These things should NOT be subclasses of Bird
- o This type of relationship is implemented by Bird containing an instance of Toy or Feathers and is called composition

How sealed Works on Classes and Methods

- o A method may be declared as sealed. This means that the method cannot be overridden in a subclass.
- o A class may also be declared as sealed. Such a class cannot be subclassed.

Why sealed?

- o Why would you make a class sealed?

Designing By Contract (DBC)

Nothing astonishes men so much as common sense and plain dealing.

- Ralph Waldo Emerson, *Essays*

- o To ensure “plain dealing” in code, we can have a contract
- o A contract documents and agrees about the rights and responsibilities of software modules
- o A contract also specifies what happens when one of the parties fails to live up to the agreement.

What methods do

- o Methods accomplish a task and before the start of the routine the method may:
 - o have an expectation about the state of the world
 - o make a statement about the state of the world when it concludes

A Method May Have

- o Preconditions: Requirements for the routine to be run
 - o It is the callers responsibility to pass good data
- o Postconditions: What the routine is guaranteed to do. The state of the world when the routine is done.
 - o Note that a postcondition implies that the routine must halt and not run forever.

Example

Pragmatic Programmer, p. 110

- o A contract for a routine that inserts a data value into a unique, ordered list.

```
public class dbc_list {
    /// <summary>
    ///
    /// <pre> contains( aNode ) == false </pre>
    /// <post>post contains( aNode ) == true </post>
    ///</summary>
    public void insertNode( Node aNode ) {
        ...
    }
}
```

Precondition Question

- o What precondition should the sqrt method have?

Interfaces: A way for supporting DBC

- o Interface: a protocol used between two unrelated entities that enables them to interact.
 - o Example: A remote control is an interface between a person and a TV
- o An interface in C#: A collection of method definitions (without implementations).
- o In Java but not C#: An interface can also include class constant declarations

What's special about interfaces?

- o An interface can only be used to specify a public interface: All interface members are (implicitly) public.
- o You can't create instances of an interface.
- o Static methods cannot be abstract, so an interface cannot contain static methods.

Why use interfaces?

- o To capture similarities among different classes without artificially forcing a class relationship.
- o Declaring methods that one of more classes are expected to implement.
- o Revealing an object's programming interface without revealing its class.

How to define an interface

- o [see interface examples]
- o Write your own implementation of Square as a shape interface and do a simple test on it