ABSTRACT CLASSES & METHODS

WARMUP

- Parent class: Shape
  - *Field: color*
  - *Parameterized constructor receives color*
  - *No methods*

- Child class: Circle
  - *Field: radius*
  - *Constructor must pass color to parent's constructor*
  - *Method: CalculateArea() should return the area of a circle (area = πr^2)*

- Child class: Rectangle
  - *Fields: width and height*
  - *Constructor must pass color to parent's constructor*
  - *Method: CalculateArea() should return the area of a rectangle*
  - *Method: Draw() that displays a rectangle based on width and height*

- Main()
  - *Create one blue Circle object with radius 4*
  - *Create one red Rectangle with width 5 and height 3.*
  - *Call CalculateArea() on both and print results.*
  - *Call Draw() on rectangle*

```
Area of circle with radius 4 is 50.24
Area of 3 x 5 rectangle is 15

Rectangle:
ooooo
ooooo
ooooo

Press any key to continue . . .
```

Now make changes!

- Add your circle and rectangle to a data structure – a List of Shape objects

- Call the same methods on the same objects, but you will need to downcast to call the methods now!

```
if(object is Dog)
{
  ((Dog)object).MethodName();
}
```

ABSTRACT CLASSES

Common issue #1 with inheritance

Perhaps.....

- All child classes should have a certain method

- I'd like the child classes to override it
  - *All child classes should have the method anyway*
  - *If I include it in the parent class, then I don't have to downcast to call the method*

- BUT… the method doesn't actually make sense to implement in the parent class
  - *I'd be writing generic code that would never actually run.*
  - *Why spend time writing code that will never be executed??*

Issue 1 - Example

```
public class Shape
{
     // What does this method actually do?
     // How big is a “shape”?  No idea.
     // But we need it here so we can override it without
     //   downcasting to call on child objects.
     public virtual double CalculateArea()
     {
           // Put some generic code in here that’s useless
           // Like:
           return 0;
     }
}

public class Rectangle: Shape
{
     // This one is much more useful and contain code that
     //   actually does something!
     public override double CalculateArea() { … }
}
```

Common issue #2 with inheritance

Perhaps.....

- My base class has things all subclasses need
  - *Reduces duplication of code*
  - *Matches up with hierarchy of objects*
- But base class doesn't "do anything" by itself
- Can I somehow prevent others from instantiating the parent class?

Issue 2 - Example

```
// This class exists to be a base class
// But it’s mostly useless by itself
public class Shape
{
}

// Squares are very useful, however
public class Square : Shape { … }

// And circles are, too.
public class Circle : Shape { … }
```

Solution: Abstract!

- We can create abstract classes and methods
  - *The two are related*
  - *Abstract methods must be in abstract classes*
- **Abstract Methods** will help with issue 1
- **Abstract Classes** will help with issue 2

Abstract methods

- Methods which have no actual code in them
  - *End the definition with a semi-colon*
  - *No { }'s*
- Defined with the **`abstract`** keyword

```
public abstract double CalcArea();
```

Abstract methods

- Abstract methods *must* be overridden in child classes – if not, the code doesn't compile!
  - *Child classes use `override` keyword*
- Sets up a rule that child classes must follow
- Any class with one or more abstract methods must itself be abstract
  - *What good is a class that has a "blank" method?*

Abstract classes

- Classes which:
  - *Can not be instantiated*
  - *Can contain zero or more abstract methods*
  - *Can also contain non-abstract (normal) methods*
  - *Any everything else normal classes can have*
- Defined with the **`abstract`** keyword

Abstract class example

```
public abstract class Shape
{
   // Fields and properties are ok!
   private String color;
   public String Color { get { return color; } }

   public Shape(String color)
   {
      this.color = color;
   }

   // Fine to mix abstract methods and normal methods!
   public abstract double CalcArea();
   public void Print() { … }
}
```

ANOTHER ABSTRACT EXAMPLE

Abstract Parent Class

```
public abstract class Animal
{
   // Animal fields
   protected string animalType;

   // Constructor as usual
   public Animal(string animalType)
   {
        this.animalType = animalType;
   }

   // Regular methods that are NOT abstract
   public void Print()
   {
      Console.WriteLine("This animal is a(n) ", animalType);
   }

   // Abstract methods
   public abstract void MakeSound();
}
```

No curly braces.

Cannot implement a block of code in an abstract method!

Non-abstract Child Classes

```
public class Cat : Animal
{
   // Cat-specific fields
   private string favoriteToy;

   // Child constructor as usual, calls parent
   public Cat(string toy) : base("cat")
   {
       this.favoriteToy = toy;
   }

   // Cverridden abstract methods
   public override void MakeSound()
   {
      Console.WriteLine("The cat meows.");
   }
}
```

Since parent class defined MakeSound( ) method as abstract, MUST override it in this child class!

HOMEWORK

HW 6&7

- Due Friday, 12/3

HW 8

- Due Saturday, 12/11 (Finals week)
- Contains lots of File IO
- Only 1 grace period may be used.
- Submissions after December 13 are NOT accepted, whether using a grace period or just late.