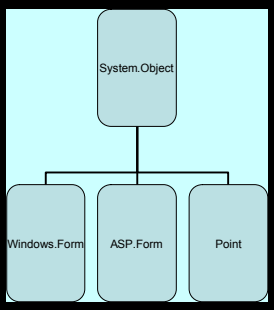
Inheritance

Class Hierarchies

- Classes do not exist in isolation. In the .NET Framework, there is a class hierarchy -- a tree of classes



Class Hierarchies

- Important part of OOP is being able to create a new class that's based on a class already defined. Defined class can be:
- a class you yourself have defined
- a standard .NET class
- a class defined by a 3rd party vendor

Derived Classes

- A class that is based on another class is called a **derived** class.
- The class it is derived from is called its base class.

Derived Classes

- Say we define a class Dog with data members name and breed.

```
public class Dog
  {
     protected string name, breed;
     public Dog() { }
  }
```

Derived Classes

- We could then derive a Shih-Tzu class from the Dog class that would inherit the data members name and breed.

Derived Classes

```
public class Shih-Tzu : Dog
{
   public Shih-Tzu() : base()
   {
   // inherits strings name, breed from Dog
        Console.WriteLine("Name is " + name);
   // add a method not true of all Dogs
   protected void BarksALot() {}
}
```

Derived Classes

- Also can inherit methods if defined correctly
- Access modifiers apply to both instance and static members..

Base Class	Derived Class
int a	Not inherited
public int b	Inherited as public int b
protected int c	Inherited as protected int c
private int d	Not inherited
internal int e	Inherited as internal int e

Constructors

- Never inherited
- Always call Base class constructor
- If not done, compiler will do it

Example

```
public class Animal
{
    private string type;  // not inherited

     public Animal(string atype)
     {
       type = atype;
     }

     public void show()
     {
       Console.WriteLine("This is a " + type;
     }

} // end Animal class
```

Example

```
public class Dog : Animal
{
    private string name, breed;

    public Dog (string aname) : base("Dog")
    {
      name = aname; breed = "Unknown";
    }

     public Dog (string aname; string abreed) : base("Dog")
     {
       name = aname; breed = abreed;
     }

} // end Dog class
```

Example

```
class TestInheritance
{

  public static void Main()
  {

      Dog dog = new Dog ("Fido", "Boxer");

      Dog nextDog = new Dog("Bonzai");

      dog.show();          // Question: What output will appear in these
      nextDog.show();   //  2 lines of code?


   } // end Main

} // end class
```

Override show()

- How can Dog override inherited show()?
- Cannot simply add a *public void show()* to Dog class.
- "hides" show in Animal

Override show()

- Make compiler happy with
- *new public void show()*

Override show()

- To override an inherited method, the base class must mark it as ***virtual*** and the derived class must mark it as ***override***

Override show()

```
public class Animal
{
  public virtual void show()
  {
    Console.WriteLine("In Animal's show");
  }
}

public class Dog : Animal
{
  public override void show()
  {
    Console. WriteLine ("In Dog's show");
  }
}
```

Override show()

```
public static void Main()
{

 Animal anim = new Dog ("Rover");

 Dog dog = new Dog ("Bonzai");

 anim.show();  // What output do we see?
 dog.show():

 } // end Main
```

Override

- When a virtual method is executed, the run-time type of the instance for which that execution takes place determines which method is called.
- In a non-virtual method invocation, the **compile-time type** of the instance is the determining factor.

Override Conditions

- All of the following must be true for a method with “override” to get a clean compile:
* An exact match can be found in some base class up that hierarchy
* Overriden method cannot be static or non-virtual (i.e., it must be virtual, abstract or override)

Override Conditions

* Base method is not sealed
* Return type is same
* Same accessibility