



COMPSCI 280 S1 2011 Enterprise Software Development

Programming Fundamentals
Inheritance



Agenda & Reading

Agenda:

- ▶ Inheritance
- ▶ Base & Derived class
- ▶ The MyBase keyword
- ▶ Constructors
- ▶ New methods
- ▶ Overriding
- ▶ Helper Methods
- ▶ Access Modifiers

Recommended Reading:

- ▶ <http://msdn.microsoft.com/en-us/library/dd460654.aspx#Inheritance>
- ▶ Microsoft Visual C# 2008 Step by Step
 - ▶ Chapter 12

Hands-On Lab:

- ▶ Lecture09Lab



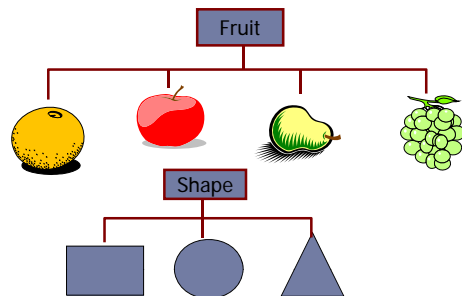
Inheritance

▶ C# supports inheritance, the ability to create a new classes from existing classes

- ▶ Base/super/parent class
- ▶ Inherited/derived/subclass/child class

▶ Relationship: is-a relationship

- ▶ An apple is a type of fruit



Person
Fields:
firstName: String
surName: String
Age: Double
...
Methods:
GetStatus: String

Common Fields and methods in base class

Employee
Fields:
Salary: Double



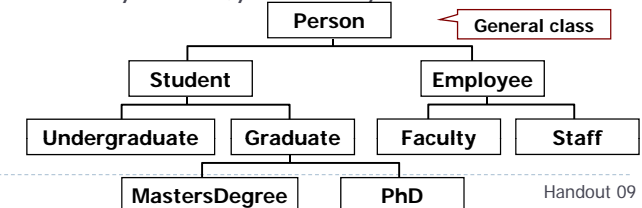
Inheritance

▶ Rules:

- ▶ Derived classes inherit, and can extend, the properties, methods, and events of the base class.
- ▶ Derived classes can also override inherited methods with new implementations.
- ▶ All classes created are inheritable by default.

▶ Characteristics:

- ▶ Inheritance allows you to write and debug a class once, and then reuse that code over and over as the basis of new classes.
- ▶ Inheritance also allows you to use inheritance-based polymorphism, the ability to define classes that can be used interchangeably by client code at run time, but with functionally different, yet identically named methods or properties.



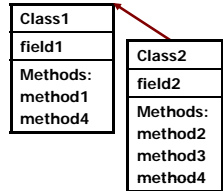


Example

- inheritance and interface implementation are defined by the : operator

```
class Class1 {
    private int field1;
    public void method1() {
        Console.WriteLine("Method1 in the base class.");
    }
    public virtual void method4(){
        Console.WriteLine("Method4 in the base class.");
    }
}

class Class2 : Class1 {
    private int field2;
    public void method2() {
        Console.WriteLine("Method2 in a derived class.");
    }
    public void method3(){
        base.method1();
        Console.WriteLine("Method3 in a derived class.");
    }
    public override void method4() {
        Console.WriteLine("Method4 in a derived class.");
    }
}
```



```
Class1 c1 = new
Class1();
Class2 c2 = new
Class2();
c1.method1();
c2.method1();
c2.method2();
c2.method3();

c1.method4();
c2.method4();
```



Constructors

- A derived class cannot inherit constructors from the base class.
- Each class must have its own constructors
 - If no constructor is defined, the compiler adds an empty constructor for the class
- Use the base keyword to make a call to the base class constructor
 - When an instance of a derived class is created, the constructor of the base class executes first, followed by constructors in derived classes.

```
class A {
    private int x;
    public A() {
        Console.WriteLine("Constructor of A");
    }
    public A(int x) {
        this.x = x;
        Console.WriteLine("A(int)");
    }
}

class B : A {
    public B() {
        Console.WriteLine("Constructor of B");
    }
    public B(int x): base(x){
        Console.WriteLine("B(int)");
    }
}

A a1 = new A();
A a2 = new A(10);
B b1 = new B();
B b2 = new B(5);
```



new methods

- If the method in the derived class is preceded with the new keyword, the method is defined as being independent of the method in the base class.
 - Using the new keyword tells the compiler that your definition hides the definition contained in the base class.

```
class A {
    ...
    public void method1(){
        Console.WriteLine("A:Method1");
    }
}

class B : A {
    public void method1(){
        Console.WriteLine("B:Method1");
    }
}

class B : A {
    public new void method1(){
        Console.WriteLine("B:Method1");
    }
}

a1.method1();
b1.method1();

A:Method1
B:Method1
```

compiler will issue a warning



Virtual methods

- If an inherited property or method needs to behave differently in the derived class it can be overridden; that is, you can define a new implementation of the method in the derived class.
 - The property or method in the base class is marked with the virtual keyword, and
 - Use the Override keyword to shadow the member by redefining it in the derived class.
- Rules:
 - The base method, which is to be overridden, must be declared as virtual, abstract, or override.
 - Both the overridden and the overriding method or property must have the same access-level modifiers
 - The new implementation of a member can call the original implementation in the parent class by specifying by the base keyword

```
class Class1 {
    ...
    public virtual void method4(){ }
}

class Class2 : Class1 {
    ...
    public override void method4() { }
}

Class1 c1 = new Class1();
Class2 c2 = new Class2();
c1.method4();
c2.method4();
```



New Vs Virtual

- ▶ The virtual keyword is needed to specify that a method should override a method of its base class.
- ▶ Attempting to override a non-virtual method will result in a compile-time error unless the "new" keyword is added to the declaration, indicating the method is intentionally hiding the base class's method
- ▶ The modifier new hides the method just like override, however there're two major differences in between
 - ▶ The original method doesn't have to be virtual
 - ▶ If you use new to hide a method and refer to the Child class using a reference of the type of Base class to point to the Child object, then call the method, you'll get the original method not the new one.

```
class A {
    public void method1(){
        Console.WriteLine("A:Method1");
    }
    public virtual void method2() {
        ...
    }
}
```

```
B b1 = new B();
b1.method1();
b1.method2();
```

B:Method1
B:Method2

Declare as A (super class)

```
class B : A {
    public new void method1(){
        Console.WriteLine("B:Method1");
    }
    public override void method2() {
        ...
    }
}
```

```
A a3 = new B();
a3.method1();
a3.method2();
```

A:Method1
B:Method2

COMPSCI280

Handout 09



Access Modifiers

- ▶ **Public**
 - ▶ The Public keyword in the declaration statement specifies that the elements are accessible from code anywhere within the same project, from other projects that reference the project, and from any assembly built from the project.
- ▶ **Protected**
 - ▶ The Protected keyword in the declaration statement specifies that the elements are accessible only from within the same class, or from a class derived from this class.
- ▶ **Private**
 - ▶ The Private keyword in the declaration statement specifies that the elements are accessible only from within the same class.



Example

- ▶ Hide
 - ▶ Instance Fields
 - ▶ Methods

```
class Super {
    public int x1 = 1;
    public int y = 1;
    public virtual void method1(){
        Console.WriteLine("Super:Method1");
    }
    public void method2() {
        Console.WriteLine("Super:Method2");
    }
    public void method3(){
        Console.WriteLine("Super:Method3");
    }
}
```

```
class Derived : Super {
    public int x2 = 2;
    public new int y = 2;
    public override void method1(){
        Console.WriteLine("Derived:method1");
    }
    public new void method2(
        Console.WriteLine("Der
    }
    public void method4(){
        Console.WriteLine("Derived:method4");
    }
    public void method5(){
        base.method3();
        Console.WriteLine("Derived:method5");
    }
}
```

Hide y from Super

Hide method2

```
Derived derived1 = new Derived();
Console.WriteLine(derived1.x1);
Console.WriteLine(derived1.x2);
Console.WriteLine(derived1.y);
derived1.method1();
derived1.method2();
derived1.method3();
derived1.method4();
derived1.method5();
```

2
Derived

```
Super derived2 = new Derived();
Console.WriteLine(derived2.x1);
Console.WriteLine(derived2.y);
derived2.method1();
derived2.method2();
derived2.method3();
```

1
Super

COMPSCI280

Handout 09



Type Casting

```
B b1 = new B();
A a3 = b1;
```

the base class reference, a3, contains a copy of the b1 reference



- ▶ **Object Conversion**
 - ▶ Make it possible to reference an object in a different way by assigning it to an object reference of a different type.
 - ▶ It is handled automatically by the compiler.
 - ▶ The object is NOT converted or changed in any way. Once instantiated, an object NEVER changes its type. The concept is similar to the "widening conversions" performed on value types.
- ▶ **Casting**
 - ▶ It is needed when the compiler doesn't recognize an automatic conversion. The concept is similar to the casting of value types to perform a "narrowing conversion".
 - ▶ Casting does not change the reference or the object being pointed to. It only changes the compiler's treatment of the reference
 - ▶ Casting is only legal between objects in the same inheritance hierarchy
 - ▶ You can safely cast from an object to its superclass



Value Types

Widening conversions

- Wider assignment
- Wider Casting

Casting a value to a wider value is always permitted but never required. However, explicitly casting can make your code more readable

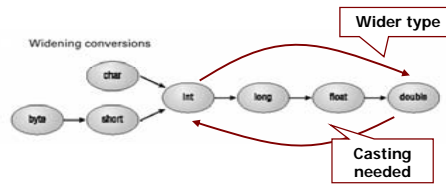
```
double d = 4.9;
int i = 10;
double d1, d2;
```

```
d1 = i;
```

Assignment conversion
d1 = 10.0

```
d2 = (double) i;
```

Casting conversion (optional)
d2 = 10.0



Narrowing conversion

- Narrow assignment – Compile-time error
- Narrow Casting – Loss of information

You must use an explicit cast to convert a value in a large type to a smaller type, or else converting that value might result in a loss of precision.

```
double d = 4.9;
int i = 10;
int i1, i2;
```

```
//i1 = d;
```

Assignment conversion
Compile-time error

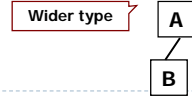
```
i2 = (int) d;
```

Narrow Casting: i2=4

NOTE: Everything beyond the decimal point will be truncated



Reference Types



Widening conversions

- Wider object reference assignment conversion
- Wider object reference casting (optional)

```
B b1 = new B();
A a3, a4;
```

```
a3 = b1;
```

```
a4 = b1 as A;
```

```
a4 = (A) b1;
```

Note:

- The as operator is like a cast except that it yields null on conversion failure instead of raising an exception
- You can safely cast from an object to its superclass
- No access to fields in subclass (e.g. a3.y)

Narrowing conversion

- Narrow object reference assignment – Compile-time error
- Narrow object reference casting – Compile-time error

Compile-time OK,

```
//b3 = a1;
```

```
A a1 = new A();
A a2 = new B();
B b3, b4, b5;
```

```
b4 = a1 as B;
```

Compile-time OK, Run-time ERROR
a1 is an instance of class A, not B

```
b5 = a2 as B;
```

Compile-time OK, Run-time OK,
a2 is an instance of B



Helper Method

The GetType method

- Gets the Type of the current instance.
- The Type instance that represents the exact runtime type of the current instance.

```
Console.WriteLine(a1.GetType());
Console.WriteLine(b1.GetType());
Console.WriteLine(a3.GetType());
Console.WriteLine(a2.GetType());
```

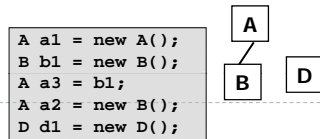
A
B
B
B

The is keyword determines whether the object is compatible with a given type.

- true if the provided expression is non-null,
- and the provided object can be cast to the provided type without causing an exception to be thrown.

```
Console.WriteLine("a1 is A=" + (a1 is A));
Console.WriteLine("b1 is A=" + (b1 is A));
Console.WriteLine("b1 is B=" + (b1 is B));
Console.WriteLine("a3 is A=" + (a3 is A));
Console.WriteLine("a3 is B=" + (a3 is B));
Console.WriteLine("b4 is A=" + (b4 is A));
Console.WriteLine("b4 is B=" + (b4 is B));
//Console.WriteLine(a3 is D);
```

a1 is A=True
b1 is A=True
b1 is B=True
a3 is A=True
a3 is B=True
b4 is A=False
b4 is B=False



Example: MyPoint class

Fields: _x, _y (private fields)

Properties X,Y (public)

- Get & Set Accessors

Methods:

- Constructors:
 - Default constructor
 - Parameterized constructor
- The ToString method
 - The Object class defines a virtual method, ToString, whose purpose is to create a string representation of the value contained in the object.
 - The default implementation simply outputs the object type.
 - All classes can override the ToString method to create their own string representation

```
MyPoint mypt1 = new MyPoint(10, 20);
MyPoint mypt2 = new MyPoint(10, 20);
MyPoint mypt4 = mypt1;
Console.WriteLine(mypt1);
```

Output: Inheritance.MyPoint
(Without overriding the ToString() method)

```
public override string ToString() {
    return "(" + _x + ", " + _y + ")";
}
```