Yin Lou
CS 2026, Spring 2010

Basic C# Features

Review

- C# types
  - Value types
    - Built-in types
    - User-defined types
    - Enumerations
  - Reference types
    - instance variables: A field declared without the static modifier is called an instance variable.
  - Boxing and unboxing
- C# arrays

Outline

- Basic C# features: OO features
  - Accessibility
  - Virtual and override
  - Class members
    - Properties
    - Indexers
    - Operator
  - Function parameters

Declared Accessibility

- Public
  - Accessible by any code in current program or other programs
- Protected
  - Accessible only by code in current class or derived classes
- Internal
  - Access limited to this program
- Protected internal
  - Accessible by code from current program or by a derived class in another program
- Private
  - Accessible only by this class

Virtual and Override

- The virtual keyword modifies methods to allow for overriding in derived classes
- By default methods are not virtual
  - You cannot override non-virtual methods
  - Unlike Java, in which methods are all virtual by default

Virtual and Override

```
public class A
{
        public virtual void F()
        {
                Console.WriteLine("Base");
        }
}

public class B: A
{
        public override void F()
        {
                base.F();
                Console.WriteLine("Derived");
        }
}

A a1 = new A(); a1.F(); //output ?
B b1 = new B(); b1.F(); //output ?
A a2 = new B(); a2.F(); //output ?
```

Class/Struct Members

- Static and instance members
- Kinds of members
  - Constants
  - Fields
  - Methods, Properties, Indexers, Operators
  - Constructors, Destructors
  - Events
  - (Nested) types

Properties

- OOP pattern in C++/Java
  - private int x;
    public int getX() { return x; }
    public void setX(int value) {x = value;}
- In C# we have elegant "properties"
  - private int x;
    public int X {
      get { return x; }
      set { x = value; }
    } // Alternatively, one can use public int X { get; set;}
  - A a = new A(); a.X = 1; int y = a.X;

Properties

- Can have three types of properties
  - Read-only: define only a get
  - Write-only: define only a set
  - Read-Write: define both get and set
- Note: fields (variables) can by read-only by using the readonly modifier

Properties

- Why properties?
  - Easy and intuitive meaning
  - Abstracts many patterns
    - Can have properties based on computation of different fields
    - e.g. Compute "age" property from date of birth
- Can be defined in interfaces
  - public int Age { get; }
  - Unlike Java

Indexers

- Special type of property
- Allows "indexing" of an object
  - bracket notation
  - E.g. hash tables: val = h[key]
    - Contrast with h.get(key)
- Syntax for declaration
  - public object this[int param1, …, int paramN]

    { get{…} set{…} }
  - Related to C++ operator[ ] overloading

Operators

- Unary
  - e.g. ++
- Binary
  - e.g. +, -, *, /
- You can overload operators to give them special meaning in your class

Operators Example

```
class A
{
        private int secret;
        public A (int val)
        {
                secret = val;
        }
        public static A operator +(A arg1, A arg2)
        {
                return new A(arg1.secret + arg2.secret);
        }
}
```

```
A var1 = new A(1);
A var2 = new A(2);
A var3 = var1 + var2;
```

Function Parameters: ref

- ref parameters
  - reference to a variable
  - can change the variable passed in

```
void F (ref int x)
{
        x = 1;
}

int x = 10;
F(ref x); // what is the value of x?
```

Function Parameters: out

- You can have functions return multiple values by using "out" parameters
- The "out" modifier is mostly like "ref"
  - out parameters must be assigned to inside the function
  - The out modifier must be used in the function definition and calling

```
void func (out value)
{ value = 1; }
int i;
func(out i);
```