Basic C# Features

Mingsheng Hong
CS 215, Spring 2008

Review

- C# types
  - Reference types
  - Value types
  - Boxing and unboxing
- C# Arrays
- First assignment released
  - Due on Feb. 1

Roadmap for Today's Lecture

- OO features
  - Accessibility
  - Virtual and override
  - Class members
    - Property
    - Indexer
    - Operator
- Function parameters

Declared Accessibility

- Public
- Protected
- Internal
  - Access limited to this program
- Protected internal
- Private

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

- Recall normal access patterns

  -
    ```
    private int x;
    public int GetX();
    public void SetX(int newVal);
    ```
  - elevated into the language:

    -
      ```
      public int X { //X is a property in class A
          get {
              return x;
          }
          set {
              x = value;
          }
      }
      ```

    -
      ```
      A a = new A();
      a.X = 1;
      int y = a.X;
      ```

Properties

- Can have three types of property
  - read-write, read-only, write-only
  - note: also have `readonly` modifier (for fields)
- Can be interface members
  `public int Age { get; };`
- Why properties?
  - abstracts many common patterns
    - static and *dynamic* properties of code
    - E.g. compute Age property from date of birth

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 string this[int a, double b]`
    `{ get{…} set{…} }`
  - Related to C++ operator[ ] overloading

Property trick for C# Arrays

- Arbitrary storage order with indexers

  -
    ```
    public int this[int a, int b] {
        get {
            // do calculation to find true location of (a,b)
            return mat[f(a, b), g(a, b)];
        }
    }
    ```

“*Any problem in computer science can be solved with another level of indirection*”,
-- Turing Award Lecture, 1993, Butler Lampson

Exercise of Indexers

- Implement a BitArray that behaves in the same way as `bool[]`, and uses 1 bit per element

Operators

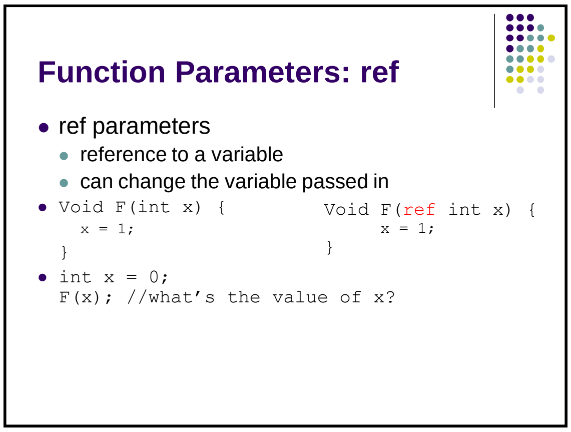
- Unary, binary, conversion

-
  ```
  class A {
      private int value;

      public A(int val)
      { value = val; }

      public static A operator +(A arg1, A arg2) {
          return new A(arg1.value + arg2.value);
      }
  }
  ```

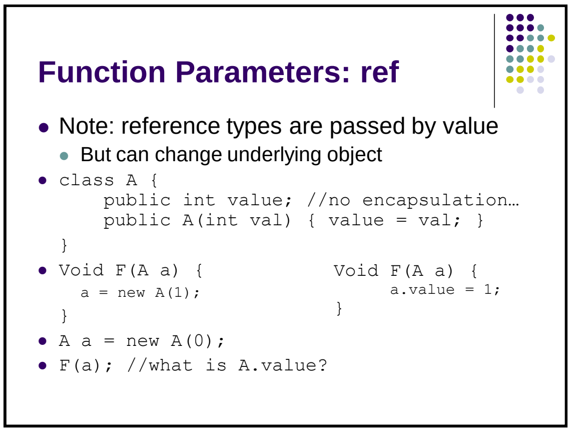
-
  ```
  A var1 = new A(1);
  A var2 = new A(2);
  A var3 = var1 + var2; //var3.value = ?
  ```



Function Parameters: ref

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

```
• Void F(int x) {                Void F(ref int x) {
     x = 1;                           x = 1;
  }                               }
• int x = 0;
  F(x); //what’s the value of x?
```



Function Parameters: ref

- Note: reference types are passed by value
  - But can change underlying object

```
• class A {
       public int value; //no encapsulation…
       public A(int val) { value = val; }
  }
• Void F(A a) {                   Void F(A a) {
     a = new A(1);                     a.value = 1;
  }                                }
• A a = new A(0);
• F(a); //what is A.value?
```