Basic C# Features

Hussam Abu-Libdeh
CS 2026, Spring 2009
Previously Discussed

• C# types
  – Reference types
  – Value types
  – Boxing and unboxing

• C# arrays

• First assignment released and emailed
  – Due on Jan 30th
Today's Agenda

- 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
• Private
  – Accessible only by this class
Declared Accessibility

- **Protected**
  - Accessible only by code in current class or derived classes

- **Internal**
  - Accessible by code in current program but not other programs
Declared Accessibility

- Protected internal
  - Accessible by code from current program or by a derived class in another program
Declared Accessibility

- Public
- Protected
- Protected internal
- Internal
- Private
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.

- `VirtualOverride.cs`
Example

- 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

```java
private int x;
public int getX() { return x; }
public void setX(int value) { x = value; }
```

- In C# we have elegant "properties"

```csharp
private int x;
public int X {
    get { return x; }
    set { x = value; }
}
```

- 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 be 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; }
  ```
Indexers

- Special type of property
- Allows index-like access of an object
  - Bracket notation
  - e.g. Hash tables: `val = h[key]`
    - Rather than `h.get(key)`
- Syntax for declaration

```csharp
public string this[ int a, double b ]
{ get {..} set {..} }
```
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
  - Pass parameters by reference
  - Can change parameter inside function

• void F (ref int x) {
  x = 1;
}

int x = 0; F(x);  // what is the value of x?

• ParametersReference.cs
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);