Writing Object Oriented Software with C#
C# and OOP

- C# is designed for the .NET Framework
  - The .NET Framework is Object Oriented
- In C#
  - Your access to the OS is through objects
  - You have the ability to create first class objects
  - The FCL is designed for extension and integration by your code
Defining Classes

class Name: BaseType{
    // Members
}

Namespace NameName{
    class Name: BaseType{
    }
}

class MyType{
    public static String someTypeState;
    public Int32 x;
    public Int32 y;
}
Accessibility

- In C#, `private` is the default accessibility

Accessibilities options
- `public` – Accessible to all
- `private` – Accessible to containing class
- `protected` – Accessible to containing or derived classes
- `internal` – Accessible to code in same assembly
- `protected internal` – means `protected` or `internal`

Classes can be marked as `public` or `internal`
- By default they are `private`
- Accessible only to code in the same source module
Type Members in C#

- **Fields**
  - The state of an object or type

- **Methods**
  - Constructors
  - Functions
  - Properties (smart fields)

- **Members come in two basic forms**
  - Instance – per object data and methods
    - Default
  - Static – per type data and methods
    - Use the `static` keyword
Methods

- Declared inline with type definition

```csharp
class MyType{
    public Int32 SomeMethod(){
        return x;
    }

    public static void StaticMethod(){
        // Do something
    }
}
```

- No inline keyword, methods are inlined when appropriate by the JIT compiler
Properties

- Methods that look like fields (smart fields)

```csharp
class Point{
    Int32 x;
    Int32 y;
    public Int32 X{
        get{return x;}
        set{x = value;}
    }
    public Int32 Y{
        get{return y;}
        set{y = value;}
    }
}
```

- Can have read-only or write-only properties
Demo Classes and Properties
Instance Constructors

- Constructors are used to initialize fields
- You can implement simpler constructors in terms of more complex ones with the **this** keyword (suggested)

```csharp
class Point{
    Int32 x;
    Int32 y;

    public Point():this(0, 0){}

    public Point(Int32 x, Int32 y){
        this.x = x;
        this.y = y;
    }
}
```

- You can indicate which base constructor to call
  - Use the **base** keyword
Type (static) Constructors

- Type constructors are used to initialize static fields for a type
- Only one static constructor per type
  - Called by the Common Language Runtime
  - Guaranteed to be called before any reference to the type or an instance of the type
  - Must have no parameters
- Use the static keyword to indicate a type constructor
Derivation and Object

- All types in the system are derived from `Object`
- You can specify a base class
  - Without a base class the compiler assumes `Object`
- Object reference variables are used as generic references
  - Collection classes in the Framework Class Library
- Object implements useful methods like
  - `ToString()`, `GetType()`
  - `ReferenceEquals()`
Polymorphism and Virtual Functions

- Use the `virtual` keyword to make a method virtual
- In derived class, override method is marked with the `override` keyword
- Example
  - `ToString()` method in Object class
    ```csharp
    public virtual string ToString();
    ```
  - Example derived class overriding `ToString()`
    ```csharp
    class SomeClass:Object{
        public override string ToString(){
            return "Some String Representing State";
        }
    }
    ```
C# and Events

- C# has built in support for events
- Great for dealing with objects in an event-driven operating system
- Improved performance and flexibility over an all-virtual-function solution
- More than one type can register interest in a single event
- A single type can register interest in any number of events
Handling an Event

EventHand.cs

```csharp
using System;
using System.Windows.Forms;
class MyForm: Form{
    MyForm()
    {
        Button button = new Button();
        button.Text = "Button";
        button.Click += new EventHandler(HandleClick);
        Controls.Add(button);
    }

    void HandleClick(Object sender, EventArgs e)
    {
        MessageBox.Show("The Click event fired!");
    }

    public static void Main()
    {
        Application.Run(new MyForm());
    }
}
```
Demo EventHand.cs
Defining an Event

- Based on a callback mechanism called a delegate

```csharp
class EventInt{
    Int32 val;
    public Int32 Value{
        get{return val;}
        set{
            if(Changed != null)
                Changed(value, val);
            val = value;
        }
    }
    public event Callback Changed;
    public delegate void Callback(Int32 newVal, Int32 oldVal);
}
```

EventInt.cs
using System;
delegate void MyDelegate(String message);
class App{
    public static void Main(){
        MyDelegate call = new MyDelegate(FirstMethod);
        call += new MyDelegate(SecondMethod);
        call("Message A");
        call("Message B");
    }
    static void FirstMethod(String str){
        Console.WriteLine("1st method: "+str);
    }
    static void SecondMethod(String str){
        Console.WriteLine("2nd method: "+str);
    }
}
Interfaces

- C# supports interfaces
  - Your types can implement interfaces
    - Must implement all methods in the interface
  - You can define custom interfaces
- Interfaces can contain methods but no fields
  - Properties and events included
  - Constructors are not supported in interfaces
- Use the `interface` keyword

```csharp
interface Name{
    // Members
}
```
Operator Overloading and Type Conversion

- C# allows you to write operator overload methods
- Called when a custom type is used in an expression with operators
  - Can overload: +, -, *, |, etc.
- Can create custom cast methods
  - Implicitly or explicitly convert your type to another type
C# and OOP

- C# and the .NET Framework promote component development
  - Can use binary or pre-compiled objects
  - More applications will use more components
  - Creates a market for third-party component vendors
  - Strong security story allows for internet deployment of objects

- C# has a great set of tools for the object oriented programmer
Writing Object Oriented Software with C#
Hidden Slides are Originals for Art in Tutorial .DOC file
Types include methods and description of fields for objects. **Stored in Exe.**

Instances are **in-memory data.** Includes data for fields and refers to type information.
A type defines a number of fields and methods. A derived type inherits the base type’s fields and methods, and adds a few of its own, to become a new type-extension of an existing type.

**Fields and Methods**
- String Description;
- FuelType Fuel;
- Double EfficiencyQuotient

**Machine**

**Fields and Methods**
- String Make;
- String Model;

**Automobile**