



Module 1 Day 9

Introduction to Classes

What makes an application?

- Program Data

- ✓ Variables & .NET Data Types
- ✓ Arrays
- ✓ More Collections (list, dictionary, stack, queue)
- ❖ Classes and objects (OOP)

- Program Logic

- ✓ Statements and expressions
- ✓ Conditional logic (if)
- ✓ Repeating logic (for, foreach, do, while)
- ✓ Methods (functions / procedures)
- ❖ Classes and objects (OOP)
- ❑ Frameworks (MVC)

- Input / Output

- User

- ✓ Console read / write
 - ❑ HTML / CSS
 - ❑ Front-end frameworks (HTML / CSS / JavaScript)

- Storage

- ❑ File I/O
 - ❑ Relational database
 - ❑ APIs

Classes

- Combine Data and Behavior to model a real-world “thing”
 - Data: Variables / properties
 - Behavior: Methods
- So far, we have used these classes (and more)
 - int, double, string, Console, Array, List, Stack, Dictionary

Classes

- Now we are going to write our own Data Types
 - These are called Classes in OO parlance
 - Remember that Classes and Types are synonymous
- e.g., Car
 - Data - describes it - adjectives
 - Make, model, color, Engine State, Gear
 - Behavior – what it can do - verbs
 - Start, Change Gear, Speed Up, Slow Down, Turn
- e.g., Contact
 - Data
 - First Name, Last Name, Birthday, Email Address, Phone
 - Behavior
 - Send Mail, Call, Text

Classes - Properties

- Automatic Properties
- Derived Properties

```
// Type (class) to represent an automobile
public class Car
{
    public string Make { get; set; }
    public string Model { get; set; }
    public int Year { get; set; }
    // A derived property for the age of the car
    public int Age
    {
        get
        {
            return DateTime.Now.Year - this.Year;
        }
    }
}
```

Classes - Properties

- Properties “backed by” a private variable (field)

```
private string gear;  
public string Gear  
{  
    get  
    {  
        return this.gear;  
    }  
    set  
    {  
        // Check to make sure the gear can be set, based on where it is now...  
        // Then, set it...  
        this.gear = value;  
    }  
}
```

Classes - Properties

- Automatic Properties
 - { get; set; }
- Derived properties
 - “getter” with no setter; the value returned comes from other state data
- Access modifiers
 - For class and for properties and variables
 - *private* and *public* (for now)
 - Users of the object can “see” public
 - Only the class itself can access private
 - We can have a **public get** with a **private set**

Classes - Methods

- Methods provide “behavior”. We’ve written lots of these.
- The **this** keyword allows access to the data held by this instance of the class

```
// Public can see the speed, but cannot set it directly
public int Speed { get; private set; }

// Accelerate 1 mph
public int Accelerate()
{
    // Check if car is in gear, then set speed
    this.Speed++;
    return this.Speed;
}
```


Classes - Constructors

- Special method that is invoked as the object is being instantiated
- Same name as the class, and NO return type
- Can accept parameters
- If you don't define one, a "default constructor" exists automatically

```
// Constructor for a Car
public Car(int year, string make, string model)
{
    this.Year = year;
    this.Make = make;
    this.Model = model;
    this.gear = "P";
}
```

Method Overloading

- Change behavior of a method based on how it is called
- Define another method:
 - With the **same name**
 - With a **different set of parameters**, as defined by their data type and order
 - Differing by parameter name only will not make it different

```
// Accelerate 1 mph
public int Accelerate()
{
    return Accelerate(1);
}

// Accelerate a certain number of mph. Can be + or -.
public int Accelerate(int amount)
{
    // check if car is in gear, then set speed
    this.Speed += amount;
    return this.Speed;
}
```

Method Overloading

A method overloaded MUST have the same name, plus:

- Overloaded methods MUST change the argument list
- Overloaded methods CAN change the return type
- Overloaded methods CAN change the access modifier