The Learning House

427 S. 4th Street #300

Louisville KY 40202

Classes in C#

.NET Cohort

Coding Bootcamp

SOFTWARE GUILD

Lesson Goals

- Understand what classes are and why we use them
- Learn how to declare classes and add Members
- Find out about different ways to access members from inside and outside the class

Review : User-Defined Types

- class
- struct
- array
- enum
- delegate
- interface

Review: What is an Object?

Objects are the key to understanding *Object-Oriented Programming*.

Just like objects in the real world, an object in code has state and behavior.

When modeling an object, ask yourself two questions:

1. What possible states can this object be in?
2. What possible behaviors can this object perform?

Example: Dog Object

State

- Name
- Color
- Breed
- IsHungry

Behavior

- Bark
- Fetch
- Eat
- Wag Tail

Example: Radio Object

State

- On
- Off
- Volume
- Station

Behavior

- TurnOn
- TurnOff
- IncreaseVolume
- DecreaseVolume
- Seek
- Scan
- Tune

Types of Class Members

Data Members Track State

- Fields
- Constants

Function Members Execute Code

- Methods
- Properties
- Constructors
- Operators
- Indexers
- Events
- Destructors

In Programmer-Speak

An object stores its state in fields (aka variables) and exposes its behavior through methods (aka functions).

Methods operate on an object's internal state and serve as a mechanism for object-to-object communication.

Hiding internal state and requiring interaction through methods is called data encapsulation.

Consider a Bicycle

State

- CurrentSpeed
- CurrentGear
- Cadence

Behavior

- ChangeGear
- Break
- ChangeCadence

By tracking state and providing methods for changing the state, we can control how the outside world is allowed to use our bicycle object.

For example, if the bike only has 6 gears, the ChangeGear method could reject any value less than 1 or greater than 6.

Objects, Objects Everywhere

We build code into individual objects for the following reasons:

1. **Modularity**: The state and behavior of an object can be written and maintained separately from other objects. Once created, we can easily pass objects as parameters and return values inside our system.
2. **Information Hiding**: Because the outside world can only interact with an object's method, the internal details of the object is hidden.
3. **Code Re-Use**: If an object already exists, you can use that object in your program. (The .NET Framework has many useful objects we can use.)
4. **Plug-ability**: Through the use of inheritance and interfaces we can easily "swap in" specialized objects to handle different cases.

Declaring a Class

In real life, you will often find many individual objects of the same type.

For example, there are thousands of bicycles in existence of the same make and model.

If each bicycle is built from the same set of blueprints and has the same components, then we say that your bicycle is an instance of the class of objects known as bicycles.

A class is the blueprint from which individual objects are created.

```
class Bicycle
{
    private int _cadence;
    private int _speed;
    private int _gear = 1;

    public void ChangeCadence(int newValue)
    {
        _cadence = newValue;
    }

    public void ChangeGear(int newValue)
    {
        if (newValue > 6 || newValue < 1)
        {
            throw new ArgumentException(
                "This bike has 6 gears", "newValue");
        }

        _gear = newValue;
    }

    public void SpeedUp(int increment)
    {
        _speed += increment;
    }

    public void ApplyBrakes(int decrement)
    {
        _speed -= decrement;
    }

    public void PrintState()
    {
        Console.WriteLine(
            "Cadence: {0}  Speed: {1}   Gear: {2}",
            _cadence, _speed, _gear);
    }
}
```

No Main()

Notice that our bicycle class does not have a main() method. This is because the class is not a complete application. It is only a blueprint for bicycles that might be used in an application.

The responsibility for creating and using bicycle objects belongs to some other class in the application.

Instantiating Objects From Classes

In order to interact with a Bicycle, we must first create an instance of a Bicycle object. We do this by using the “new” keyword. Invoking new constructs a new object.

Once we have an instance, we can then invoke its methods.

What will the output of this code be?

cadence: 50 speed:10 gear:2
cadence: 40 speed:20 gear:3

```
static void Main()
{
    // create two bicycle objects
    Bicycle bike1 = new Bicycle();
    Bicycle bike2 = new Bicycle();

    // Invoke methods on the objects
    bike1.ChangeCadence(50);
    bike1.SpeedUp(10);
    bike1.ChangeGear(2);
    bike1.PrintState();

    bike2.ChangeCadence(50);
    bike2.SpeedUp(10);
    bike2.ChangeGear(2);
    bike2.ChangeCadence(40);
    bike2.SpeedUp(10);
    bike2.ChangeGear(3);
    bike2.PrintState();
}
```

Data Members

Also known as fields. These are variables that belong to a class.

- Fields can be of any type, even user-defined types like other classes.
- Like variables, fields store data and can be read from and written to.
- Most often, fields are used to store information that needs to be available to the whole class.
- We don't usually make fields public (accessible to other classes). Instead, if we want other classes to see data, we generally use a property (coming later).

```
class Bicycle
{
    private int _cadence;
    private int _speed;
    private int _gear = 1;
```

Field Initializers

Class fields are automatically initialized to the default for their type when an object is instantiated.

We can override the default by giving a default value.

```
class Bicycle
{
    private int _cadence;  // 0
    private int _speed;    // 0
    private int _gear = 1; // 1
```

Access Modifiers

Members of a class can be prefixed with an access modifier. Inside a class instance, any function member or data member can access all the other members.

Sometimes we want to share data or function members with other classes and even other developer's programs.

Access modifiers specify the limitations on who can call a data or function member.

Five Access Modifiers

- **private** – only members inside the same object can access them.
- **public** – any other object in any program may access it.
- **protected**
- **internal**
- **protected internal**

(we will cover these later)

Public vs. Private

Notice that the Bicycle class has a private field _cadence and a public method ChangeCadence().

The ChangeCadence() method can use the private field because they are declared in the same class.

However, when we create an instance of the Bicycle object in Main(), we can only use the ChangeCadence() method. We can not access the _cadence field directly because it is private.

```
class Bicycle
{
    private int _cadence;

    public void ChangeCadence(int newValue)
    {
        _cadence = newValue;
    }
```

```
 static void Main()
 {
     Bicycle bike1 = new Bicycle();

     bike1.ChangeCadence(20);
     bike1._cadence = 50;
```

Cannot access private field '_cadence' here

Build a class model for a smartphone address book

LAB EXERCISE

SOFTWARE GUILD