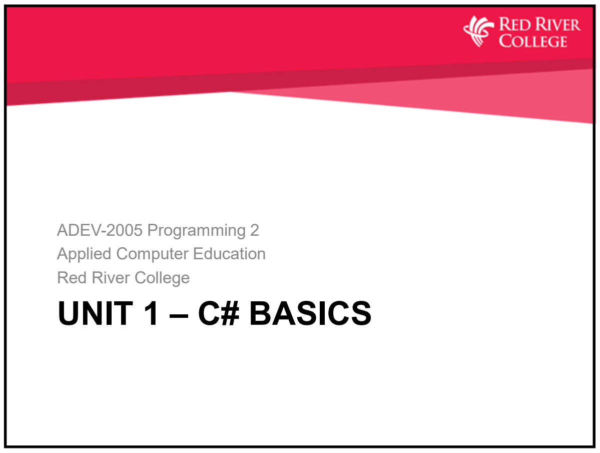
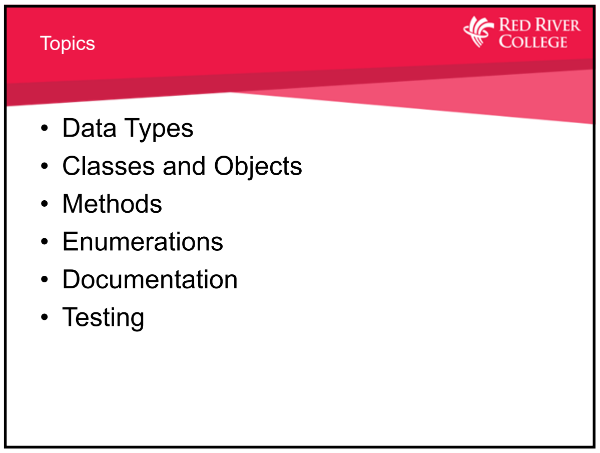
RED RIVER COLLEGE
OF APPLIED ARTS, SCIENCE AND TECHNOLOGY



ADEV-2005 Programming 2
Applied Computer Education
Red River College

UNIT 1 – C# BASICS



Topics

- Data Types
- Classes and Objects
- Methods
- Enumerations
- Documentation
- Testing

Primitive Data Types

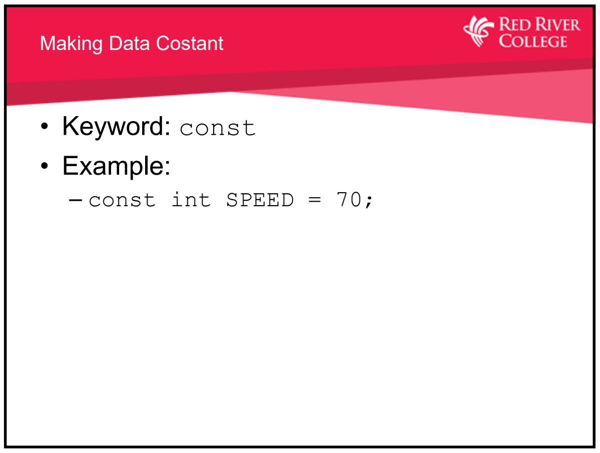
C# Type	Type	.NET alias type
char	Integral	System.Char
double	Floating-point	System.Double
int	Integral	System.Int32
bool	Boolean	System.Boolean

Decimal Type

- New to modern languages
- Used for monetary data
  - Allows integral and fractions
  - Greater precision (128 bit)
- .NET data type: System.Decimal

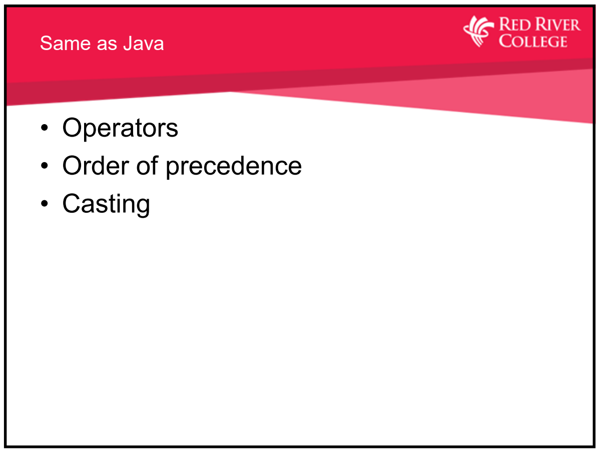
String Data Type

- Build-in type
- Keyword: `string`
- .NET data type: System.String
- Examples:
  - `string courseName = “Programming 2”;`
  - `string twoLines = “Line 1\nLine 2”;`
  - `string path = @”C:\User\TC\project”;`



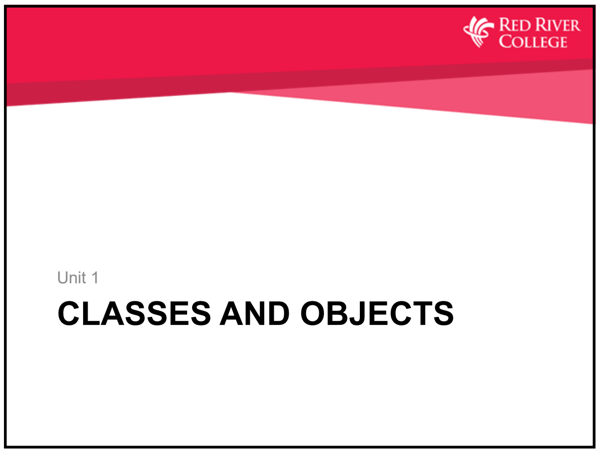
Making Data Costant

- Keyword: `const`
- Example:
  - `const int SPEED = 70;`



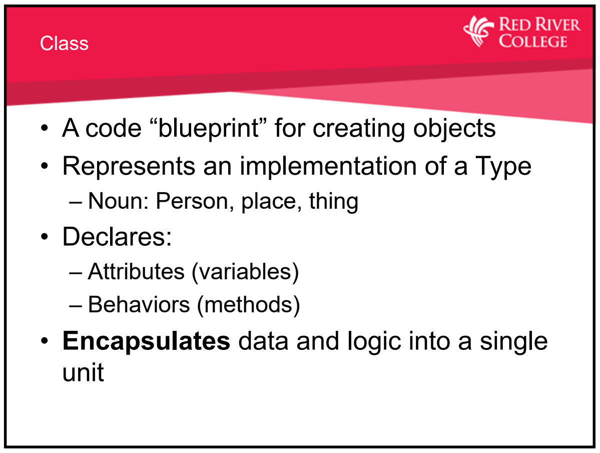
Same as Java

- Operators
- Order of precedence
- Casting



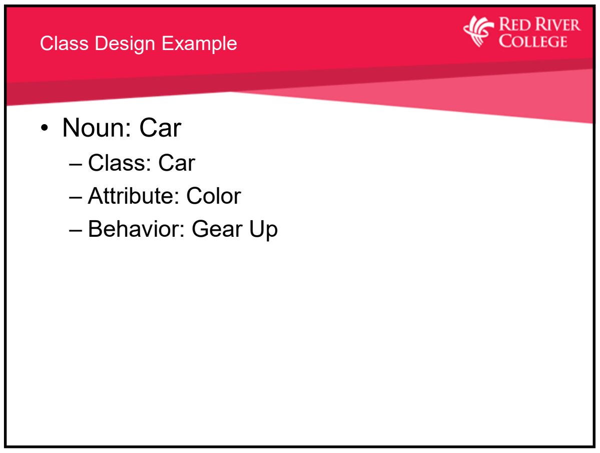
Unit 1

CLASSES AND OBJECTS



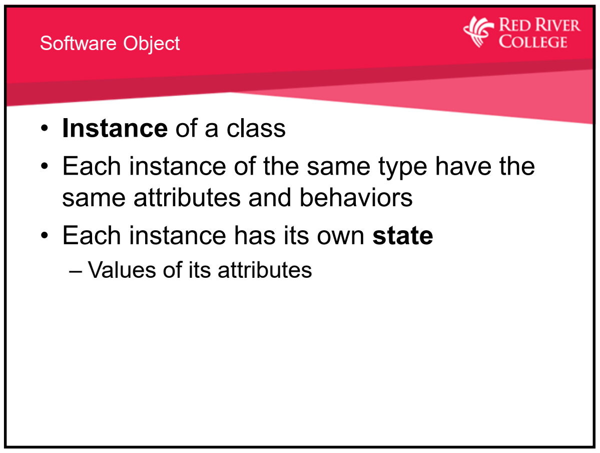
Class

- A code “blueprint” for creating objects
- Represents an implementation of a Type
  - Noun: Person, place, thing
- Declares:
  - Attributes (variables)
  - Behaviors (methods)
- **Encapsulates** data and logic into a single unit



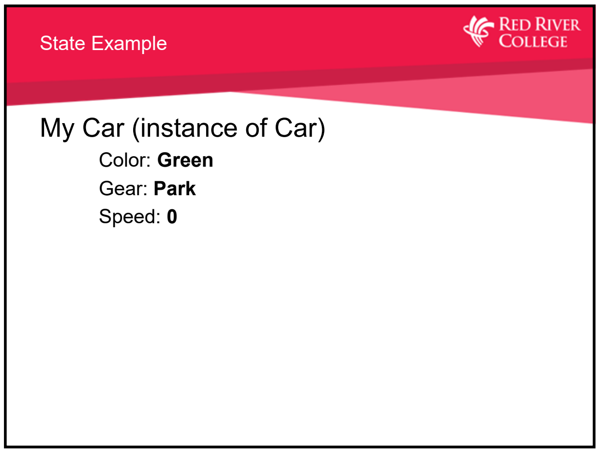
Class Design Example

- Noun: Car
  - Class: Car
  - Attribute: Color
  - Behavior: Gear Up



Software Object

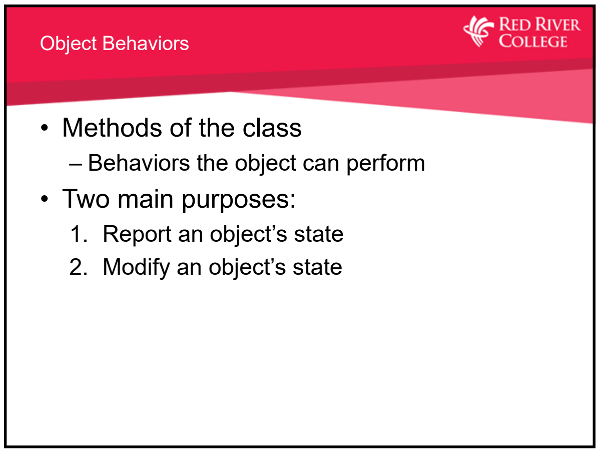
- **Instance** of a class
- Each instance of the same type have the same attributes and behaviors
- Each instance has its own **state**
  - Values of its attributes



State Example

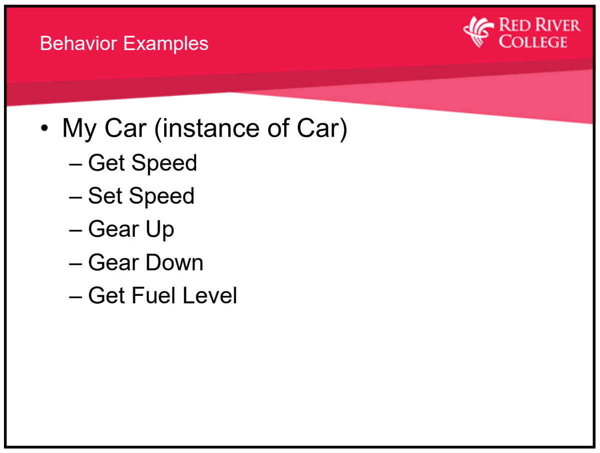
My Car (instance of Car)

Color: **Green**
Gear: **Park**
Speed: **0**



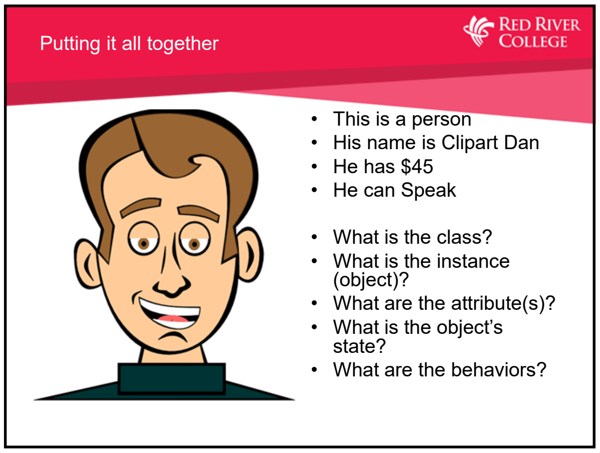
Object Behaviors

- Methods of the class
  - Behaviors the object can perform
- Two main purposes:
  1. Report an object's state
  2. Modify an object's state



Behavior Examples

- My Car (instance of Car)
  - Get Speed
  - Set Speed
  - Gear Up
  - Gear Down
  - Get Fuel Level

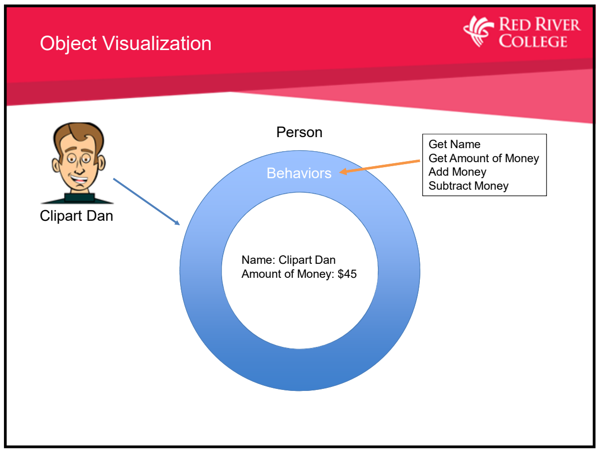


Putting it all together

RED RIVER COLLEGE

- This is a person
- His name is Clipart Dan
- He has $45
- He can Speak

- What is the class?
- What is the instance (object)?
- What are the attribute(s)?
- What is the object's state?
- What are the behaviors?



Object Visualization

RED RIVER COLLEGE

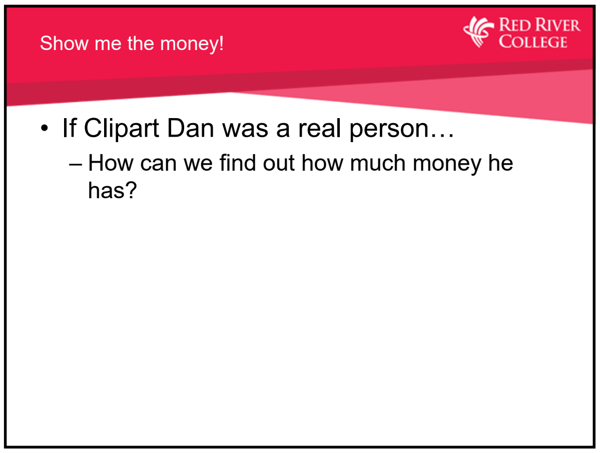
Clipart Dan

Person

Behaviors

Get Name
Get Amount of Money
Add Money
Subtract Money

Name: Clipart Dan
Amount of Money: $45

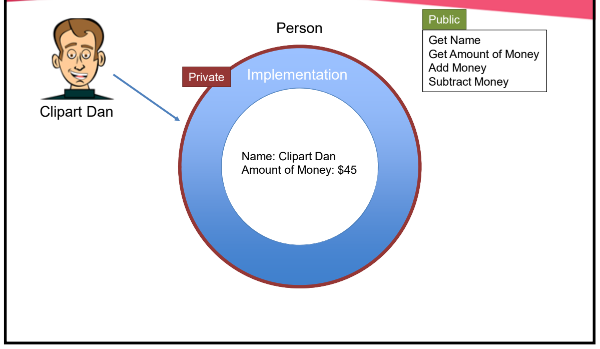


Show me the money!

RED RIVER COLLEGE

- If Clipart Dan was a real person…
  - How can we find out how much money he has?

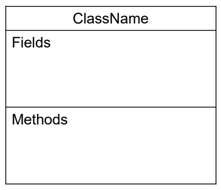
Encapsulation

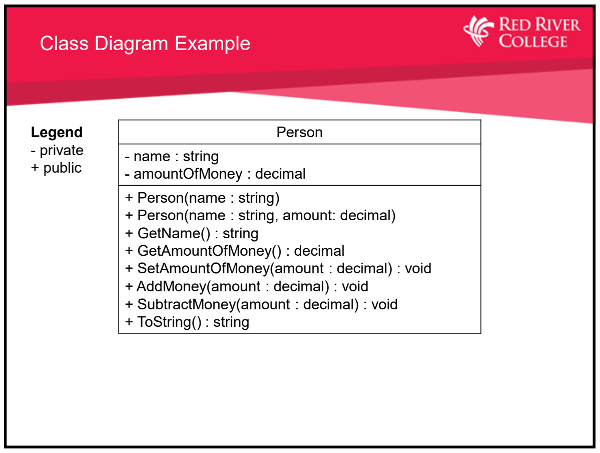


Members of a Class

- Attributes
  - Fields (class scope variables)
- Behaviors
  - Methods
- Constructors
  - Methods for setting initial state of an instance

Class Diagram

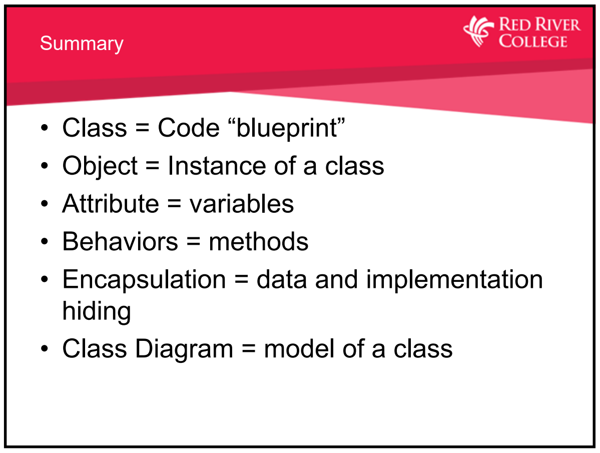




Class Diagram Example

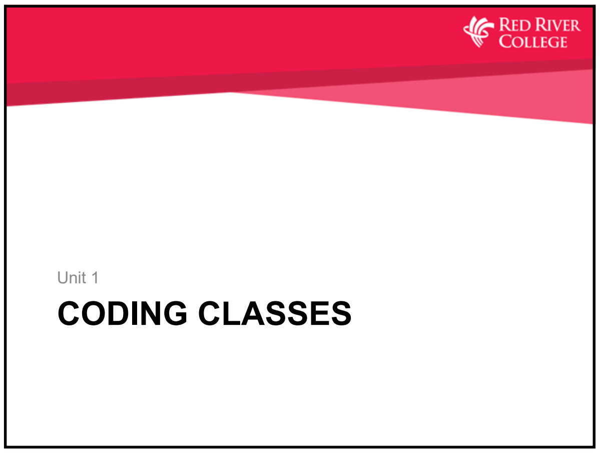
Legend
- private
+ public

Person
- name : string - amountOfMoney : decimal
+ Person(name : string) + Person(name : string, amount: decimal) + GetName() : string + GetAmountOfMoney() : decimal + SetAmountOfMoney(amount : decimal) : void + AddMoney(amount : decimal) : void + SubtractMoney(amount : decimal) : void + ToString() : string



Summary

- Class = Code "blueprint"
- Object = Instance of a class
- Attribute = variables
- Behaviors = methods
- Encapsulation = data and implementation hiding
- Class Diagram = model of a class

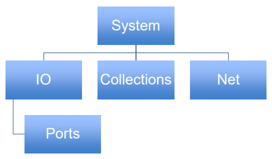


Unit 1

CODING CLASSES

Namespaces

- An identifier
- Organizes code (ex. classes) into logical groups
- Defines a scope
- Prevents name collision

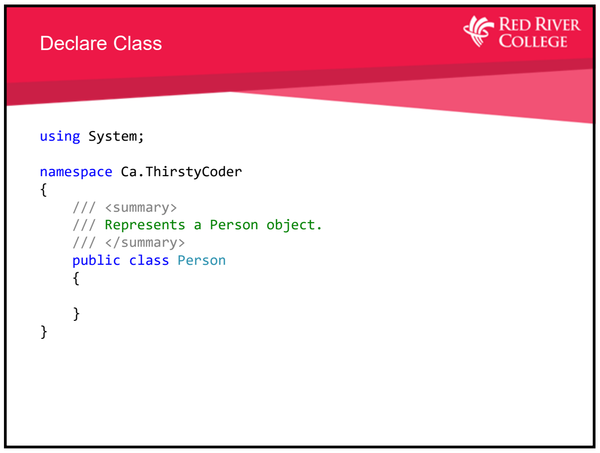


Declaring Namespace Syntax

- Syntax
  - `namespace RootNamespace[.Sub.Sub.Sub]`
- Example Identifiers:
  - System
  - System.IO
  - Ca.Rrc.StudentServices

Declaring Namespace Example

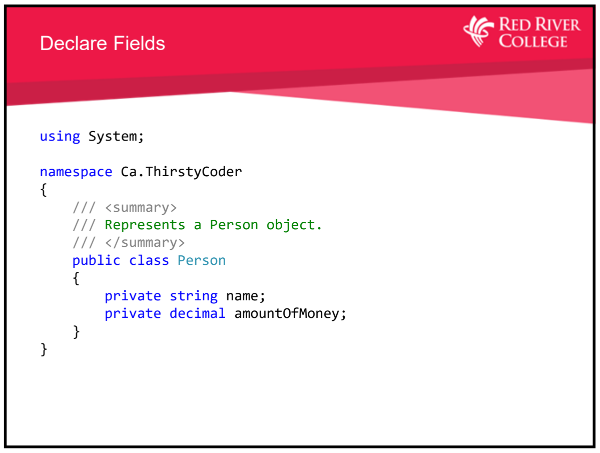
```
using System;

namespace Ca.ThirstyCoder
{

}
```



Declare Class

RED RIVER COLLEGE

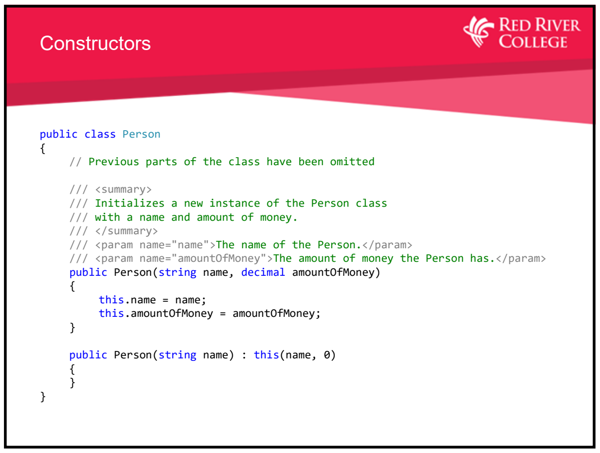
```
using System;

namespace Ca.ThirstyCoder
{
    /// <summary>
    /// Represents a Person object.
    /// </summary>
    public class Person
    {

    }
}
```



Declare Fields

RED RIVER COLLEGE

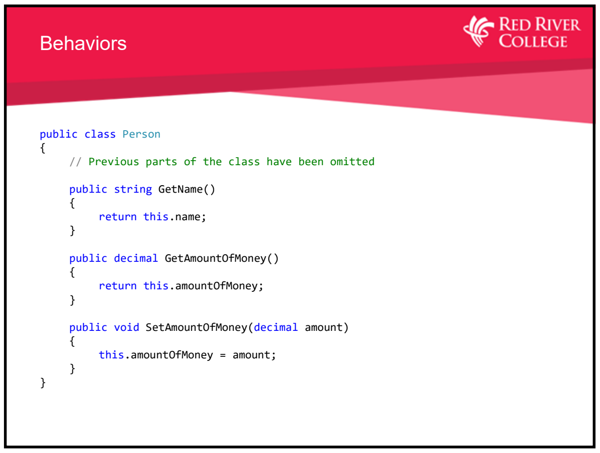
```
using System;

namespace Ca.ThirstyCoder
{
    /// <summary>
    /// Represents a Person object.
    /// </summary>
    public class Person
    {
        private string name;
        private decimal amountOfMoney;
    }
}
```



Constructors

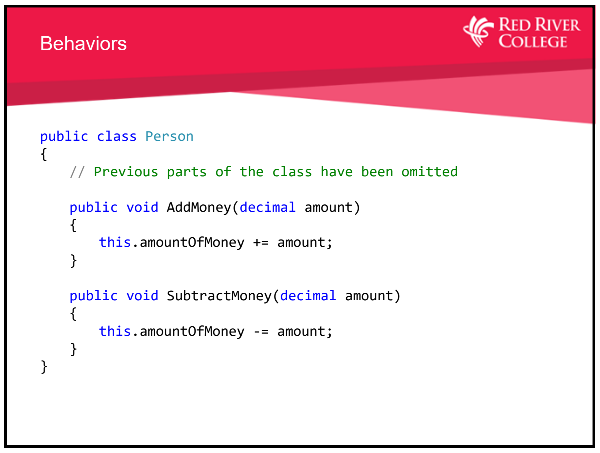
RED RIVER COLLEGE

```
public class Person
{
    // Previous parts of the class have been omitted

    /// <summary>
    /// Initializes a new instance of the Person class
    /// with a name and amount of money.
    /// </summary>
    /// <param name="name">The name of the Person.</param>
    /// <param name="amountOfMoney">The amount of money the Person has.</param>
    public Person(string name, decimal amountOfMoney)
    {
        this.name = name;
        this.amountOfMoney = amountOfMoney;
    }

    public Person(string name) : this(name, 0)
    {
    }
}
```



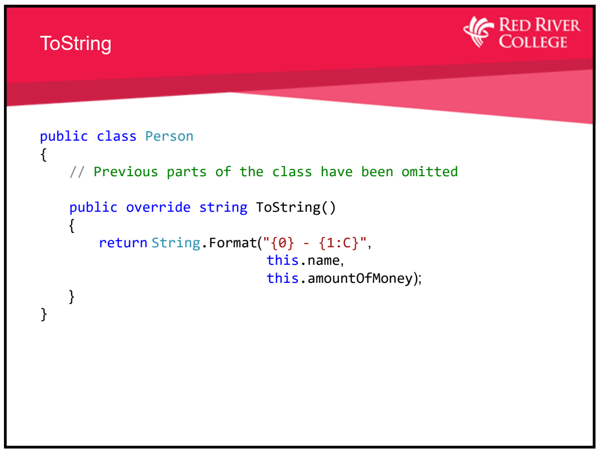
Behaviors

```
public class Person
{
    // Previous parts of the class have been omitted

    public string GetName()
    {
        return this.name;
    }

    public decimal GetAmountOfMoney()
    {
        return this.amountOfMoney;
    }

    public void SetAmountOfMoney(decimal amount)
    {
        this.amountOfMoney = amount;
    }
}
```



Behaviors

```
public class Person
{
    // Previous parts of the class have been omitted

    public void AddMoney(decimal amount)
    {
        this.amountOfMoney += amount;
    }

    public void SubtractMoney(decimal amount)
    {
        this.amountOfMoney -= amount;
    }
}
```



ToString

```
public class Person
{
    // Previous parts of the class have been omitted

    public override string ToString()
    {
        return String.Format("{0} - {1:C}",
                             this.name,
                             this.amountOfMoney);
    }
}
```

Instance

- The realization of an object from a class
- Each instance of a type has…
  - identical attributes
  - identical behaviors
  - its own state

Object Construction

- **new** keyword
- Followed by the invocation of a class' **constructor** method

Object Construction Examples

```
static void main(string[] args)
{
    Person damien;

    damien = new Person("Damien");

    Person clipartDan = new Person("Clipart Dan", 45);

    Console.Write("Press any key to continue...");
    Console.ReadKey();
}
```

Black Box

- Computer science term
- Input something into the box
- Produces output from the box
- Without knowledge of its internal workings
- A class is the blueprint for constructing a box (internal workings)
- An object is the box (instance from the blueprint)

Object Interface

- The operations that can be invoked on the object
  - Often to report and update object's state
- public (accessible) members of the class
- "inputs" - parameters
- "outputs" – return value

Object Interface Example

Class Documentation

- Manual of an object's interface
- Includes:
  - Overall purpose for the type
  - How instances are constructed
  - Explanation of operations it can perform (methods)
    - "inputs" – parameters
    - "outputs" – return values

Accessing Interface

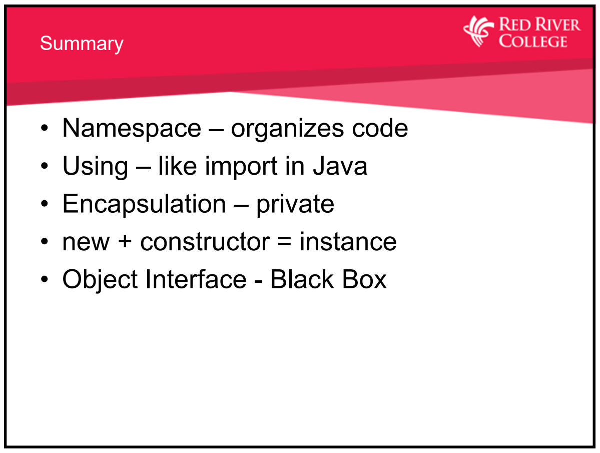
```
static void main(string[] args)
{
     Person damien;

     damien = new Person("Damien");

     Person clipartDan = new Person("Clipart Dan", 45);

     damien.SetAmountOfMoney(100);

     clipartDan.AddMoney(13.45M);

     clipartDan.SubtractMoney(7.77M);

     Console.WriteLine("{0}: {1:C}", damien.GetName(), damien.GetAmountOfMoney());
     Console.WriteLine(clipartDan);

     Console.Write("Press any key to continue...");
     Console.ReadKey();
}
```

API Developer

- Develops classes (framework) for completing application tasks
- Develops documentation (manual) for working with objects of the defined types

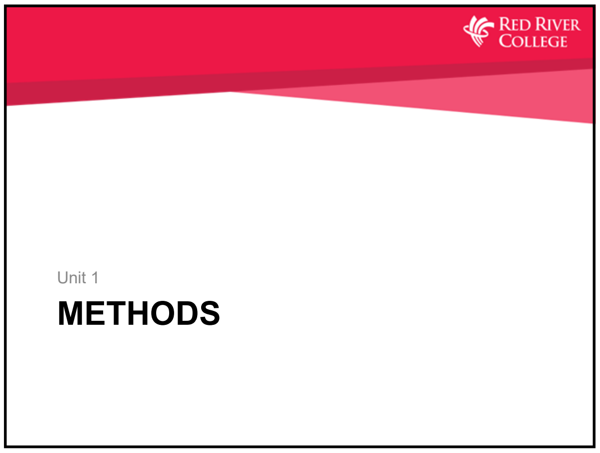
Application Developer

- Develops user interface (UI)
- Uses framework (API) to complete application tasks



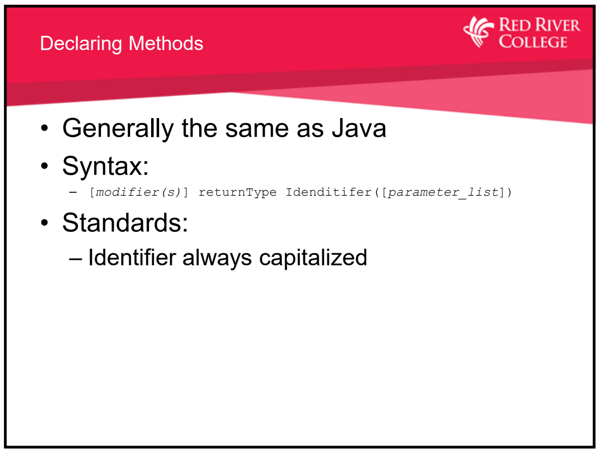
Summary

- Namespace – organizes code
- Using – like import in Java
- Encapsulation – private
- new + constructor = instance
- Object Interface - Black Box



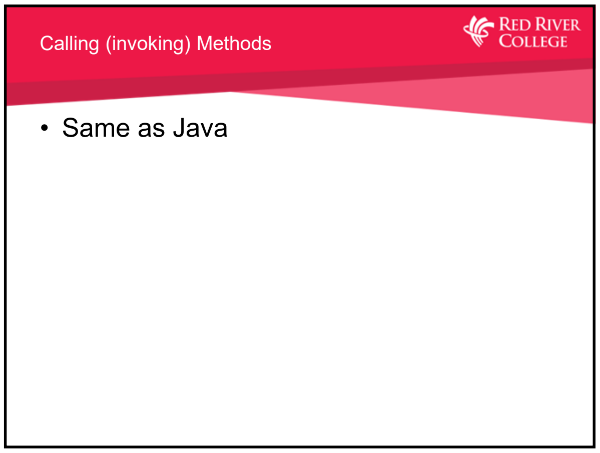
Unit 1

METHODS



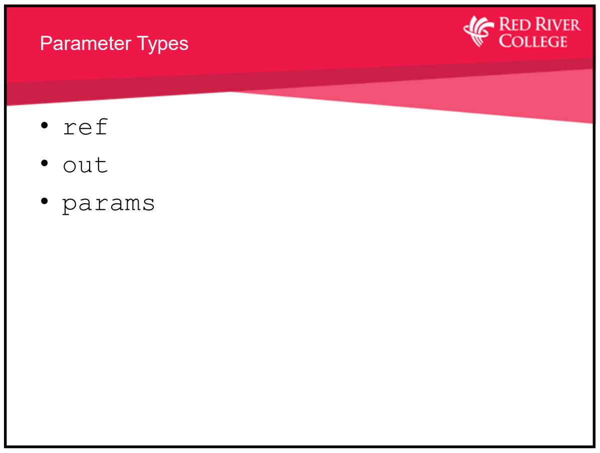
Declaring Methods

- Generally the same as Java
- Syntax:
  - `[modifier(s)] returnType Idenditifer([parameter_list])`
- Standards:
  - Identifier always capitalized



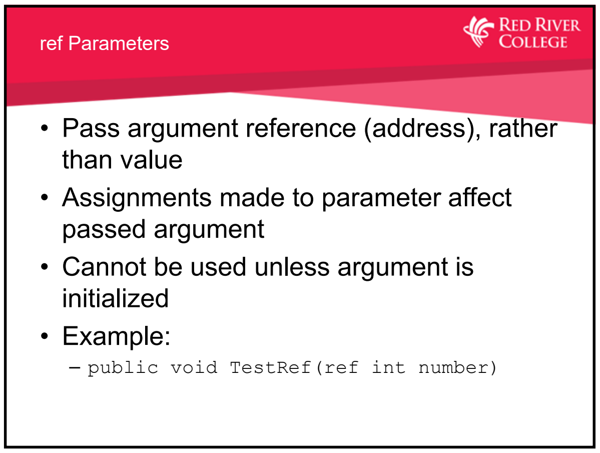
Calling (invoking) Methods

- Same as Java



Parameter Types

- `ref`
- `out`
- `params`



ref Parameters

- Pass argument reference (address), rather than value
- Assignments made to parameter affect passed argument
- Cannot be used unless argument is initialized
- Example:
  - `public void TestRef(ref int number)`

ref Example

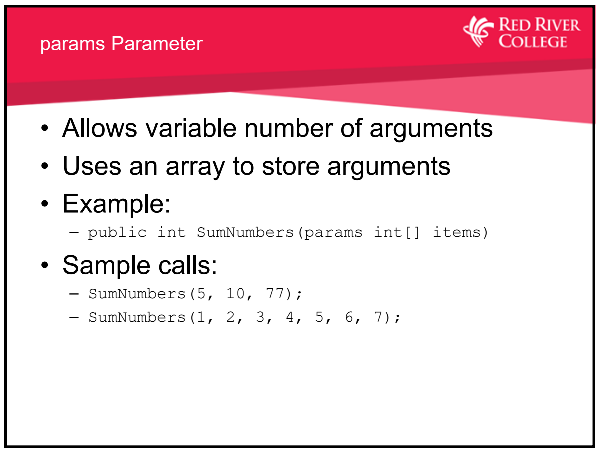
```
public void TestRef(ref int number)
{
    number = 99;
}

static void Main(string[] args)
{
    int age = 5;
    TestRef(ref age);
    // age now equals 99 at this point
}
```

out Parameter

- Pass argument reference (address), rather than value
- Assignments made to parameter affect passed argument
- Argument does not need to be initialized
- Example:
  - `public void TestOut(out int number)`

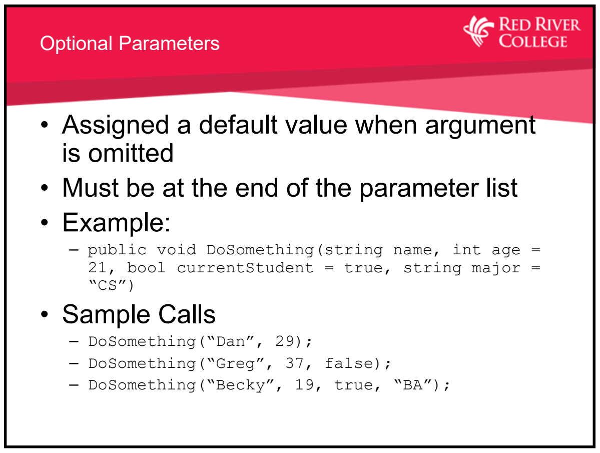
out Example

```
public void TestOut(out int number)
{
    number = 99;
}

static void Main(string[] args)
{
    int age;
    TestOut(out age);
    // age now equals 99 at this point
}
```



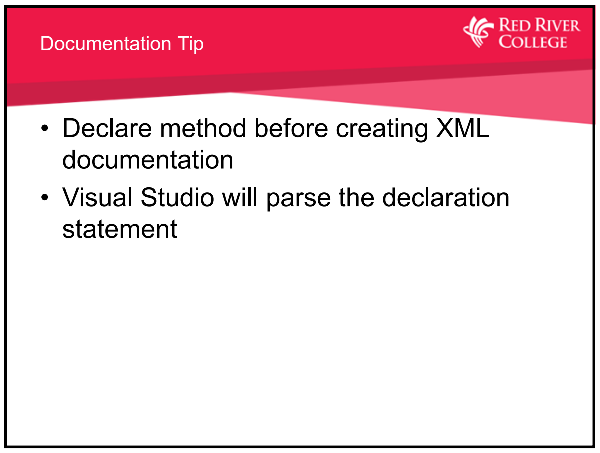
params Parameter

- Allows variable number of arguments
- Uses an array to store arguments
- Example:
  - `public int SumNumbers(params int[] items)`
- Sample calls:
  - `SumNumbers(5, 10, 77);`
  - `SumNumbers(1, 2, 3, 4, 5, 6, 7);`



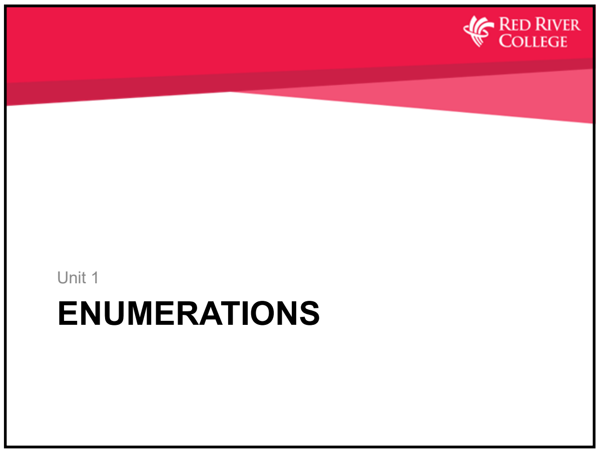
Optional Parameters

- Assigned a default value when argument is omitted
- Must be at the end of the parameter list
- Example:
  - `public void DoSomething(string name, int age = 21, bool currentStudent = true, string major = “CS”)`
- Sample Calls
  - `DoSomething(“Dan”, 29);`
  - `DoSomething(“Greg”, 37, false);`
  - `DoSomething(“Becky”, 19, true, “BA”);`



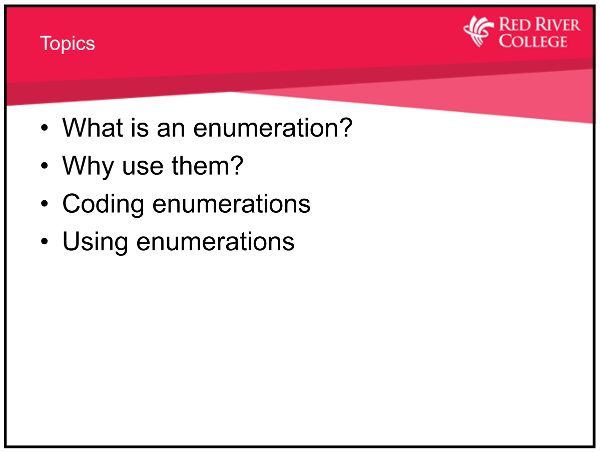
Documentation Tip

- Declare method before creating XML documentation
- Visual Studio will parse the declaration statement



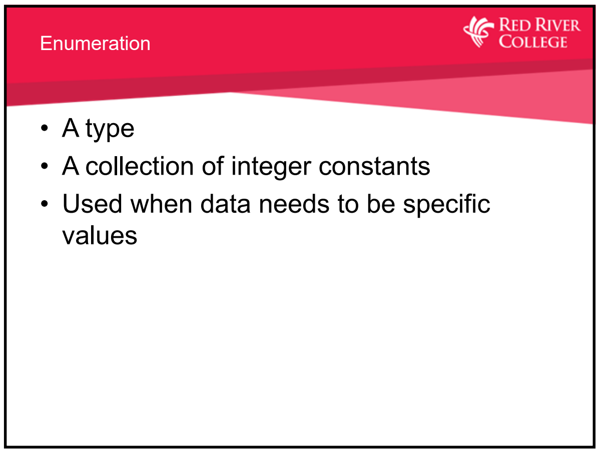
Unit 1

ENUMERATIONS



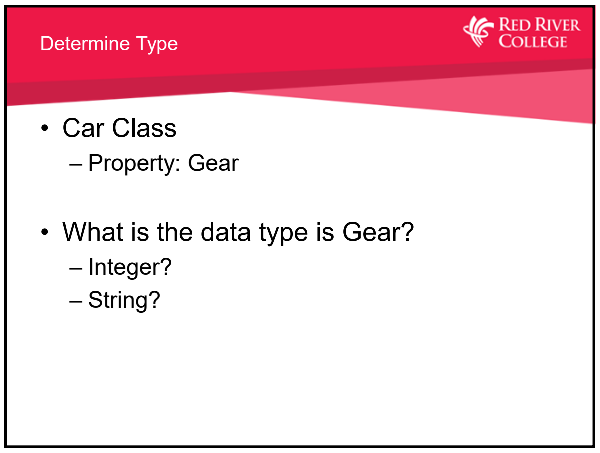
Topics

- What is an enumeration?
- Why use them?
- Coding enumerations
- Using enumerations



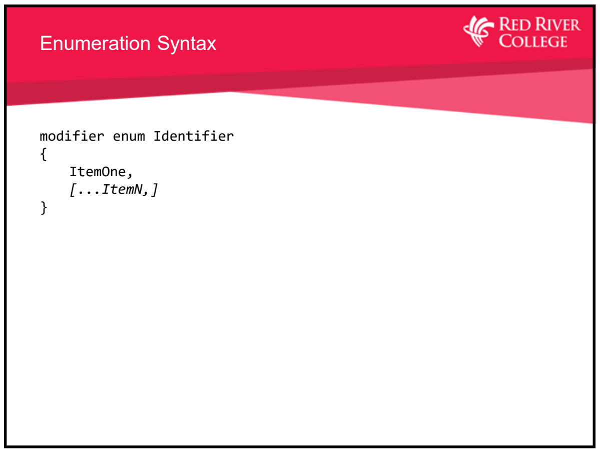
Enumeration

- A type
- A collection of integer constants
- Used when data needs to be specific values



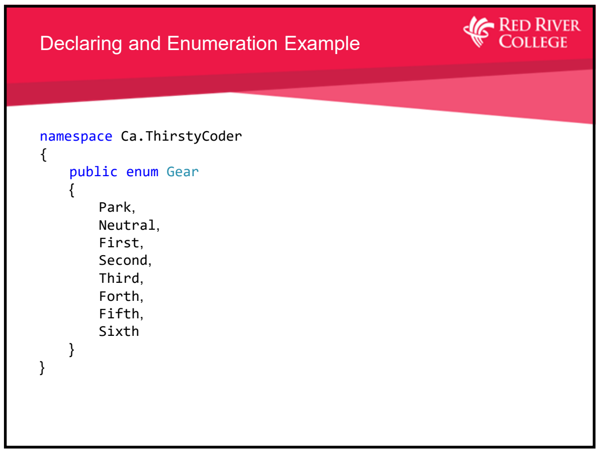
Determine Type

- Car Class
  - Property: Gear
- What is the data type is Gear?
  - Integer?
  - String?



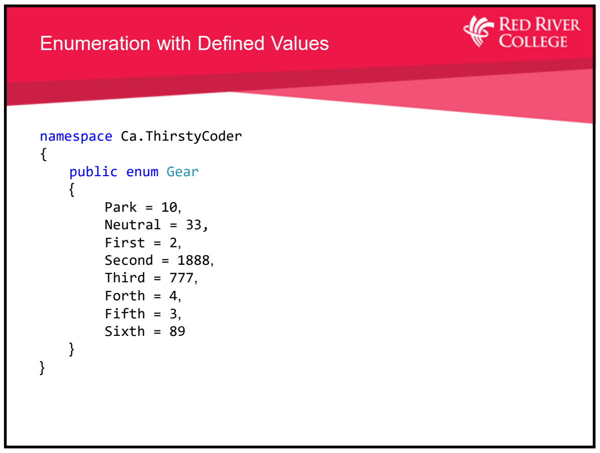
Enumeration Syntax

```
modifier enum Identifier
{
    ItemOne,
    [...ItemN,]
}
```



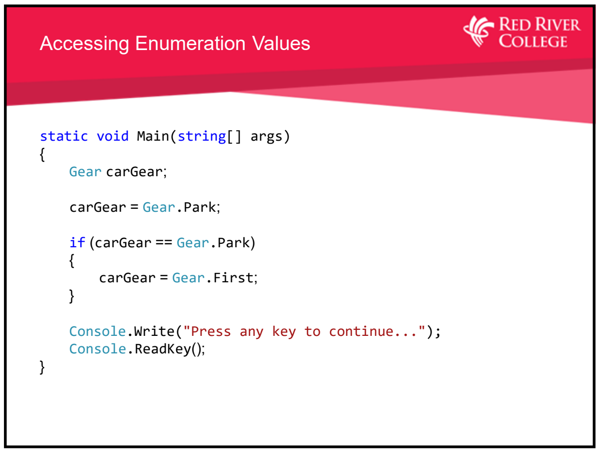
Declaring and Enumeration Example

```
namespace Ca.ThirstyCoder
{
    public enum Gear
    {
        Park,
        Neutral,
        First,
        Second,
        Third,
        Forth,
        Fifth,
        Sixth
    }
}
```



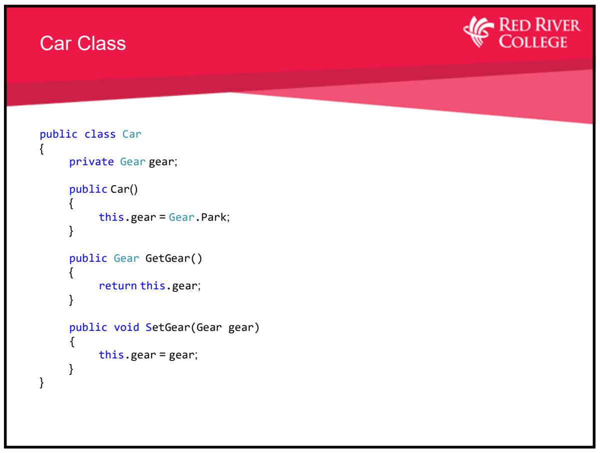
Enumeration with Defined Values

```
namespace Ca.ThirstyCoder
{
    public enum Gear
    {
        Park = 10,
        Neutral = 33,
        First = 2,
        Second = 1888,
        Third = 777,
        Forth = 4,
        Fifth = 3,
        Sixth = 89
    }
}
```



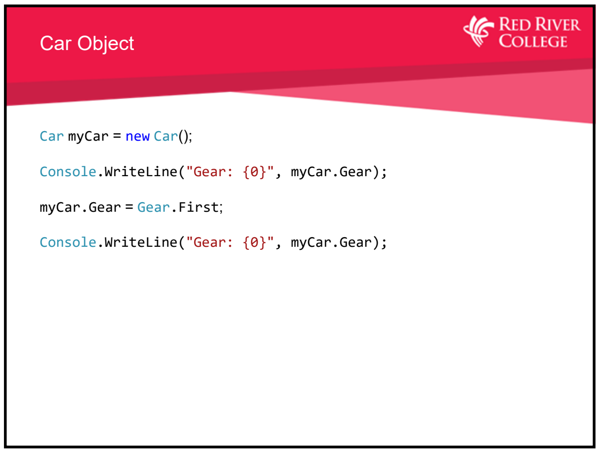
Accessing Enumeration Values

```
static void Main(string[] args)
{
    Gear carGear;

    carGear = Gear.Park;

    if (carGear == Gear.Park)
    {
        carGear = Gear.First;
    }

    Console.Write("Press any key to continue...");
    Console.ReadKey();
}
```



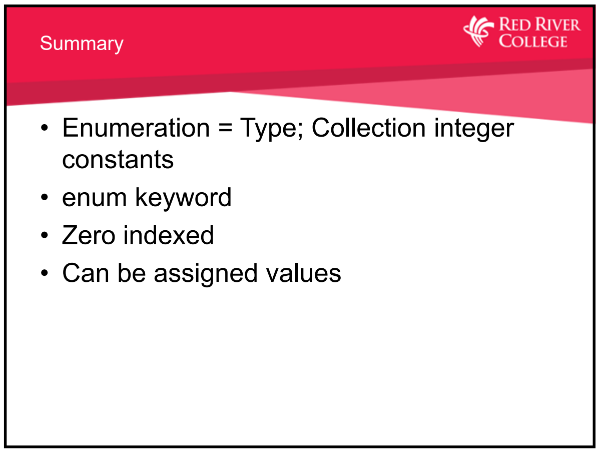
Car Class

```
public class Car
{
    private Gear gear;

    public Car()
    {
        this.gear = Gear.Park;
    }

    public Gear GetGear()
    {
        return this.gear;
    }

    public void SetGear(Gear gear)
    {
        this.gear = gear;
    }
}
```



Car Object

```
Car myCar = new Car();
Console.WriteLine("Gear: {0}", myCar.Gear);
myCar.Gear = Gear.First;
Console.WriteLine("Gear: {0}", myCar.Gear);
```



Summary

- Enumeration = Type; Collection integer constants
- enum keyword
- Zero indexed
- Can be assigned values