Classes

Object-Oriented Programming

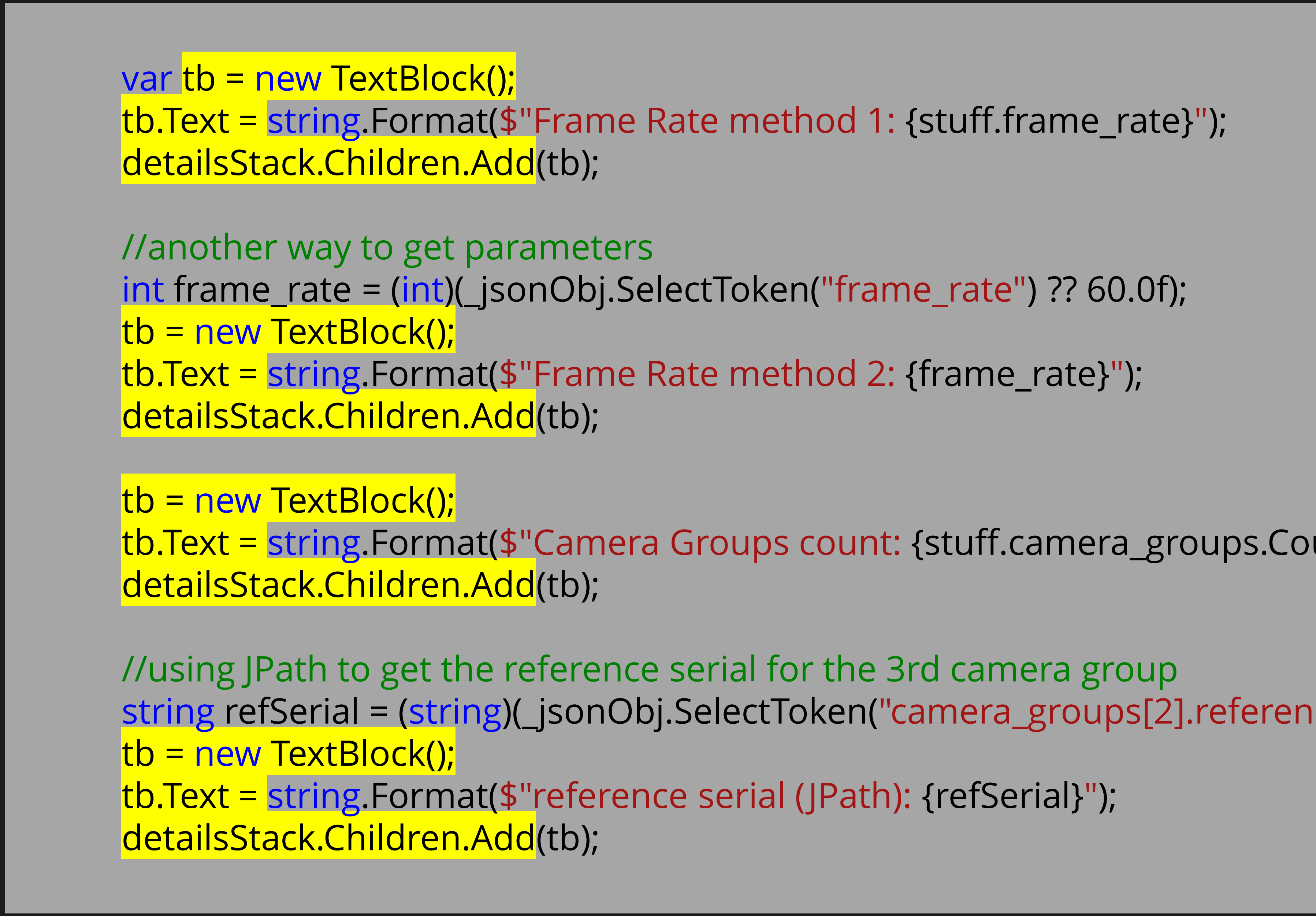
Topics

- D.R.Y.
- Object-Oriented Programming
- Class Libraries
- Fields
- Properties
- Constructors
- Methods
- Instances
- Interfaces

Essential Skill: D.R.Y.

- **D**on’t **R**epeat **Y**ourself

Why is it bad to repeat code?



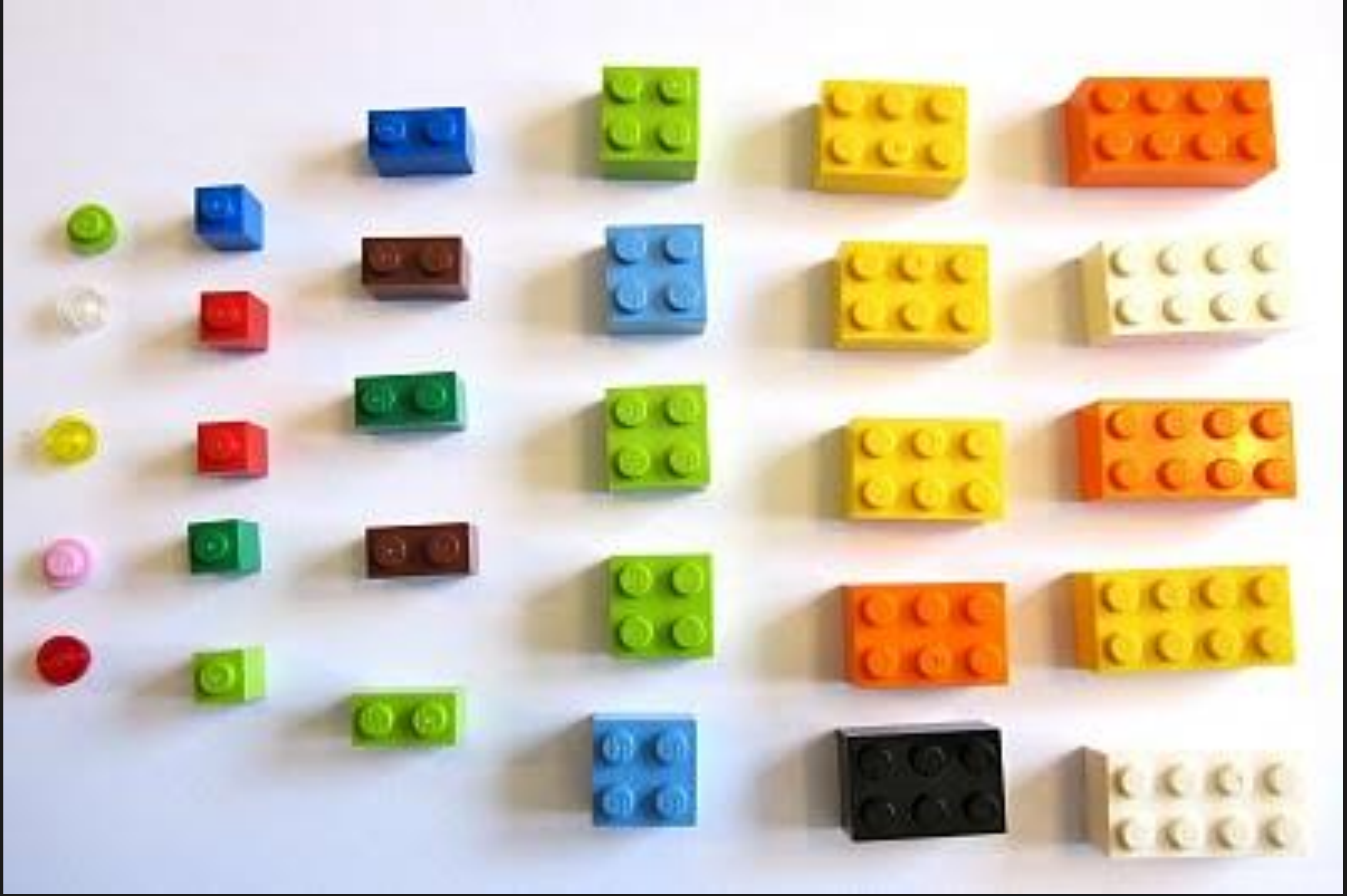
```
var tb = new TextBlock();
tb.Text = string.Format($"Frame Rate method 1: {stuff.frame_rate}");
detailsStack.Children.Add(tb);

//another way to get parameters
int frame_rate = (int)(_jsonObj.SelectToken("frame_rate") ?? 60.0f);
tb = new TextBlock();
tb.Text = string.Format($"Frame Rate method 2: {frame_rate}");
detailsStack.Children.Add(tb);

tb = new TextBlock();
tb.Text = string.Format($"Camera Groups count: {stuff.camera_groups.Co
detailsStack.Children.Add(tb);

//using JPath to get the reference serial for the 3rd camera group
string refSerial = (string)(_jsonObj.SelectToken("camera_groups[2].referen
tb = new TextBlock();
tb.Text = string.Format($"reference serial (JPath): {refSerial}");
detailsStack.Children.Add(tb);
```

Object-Oriented Programming

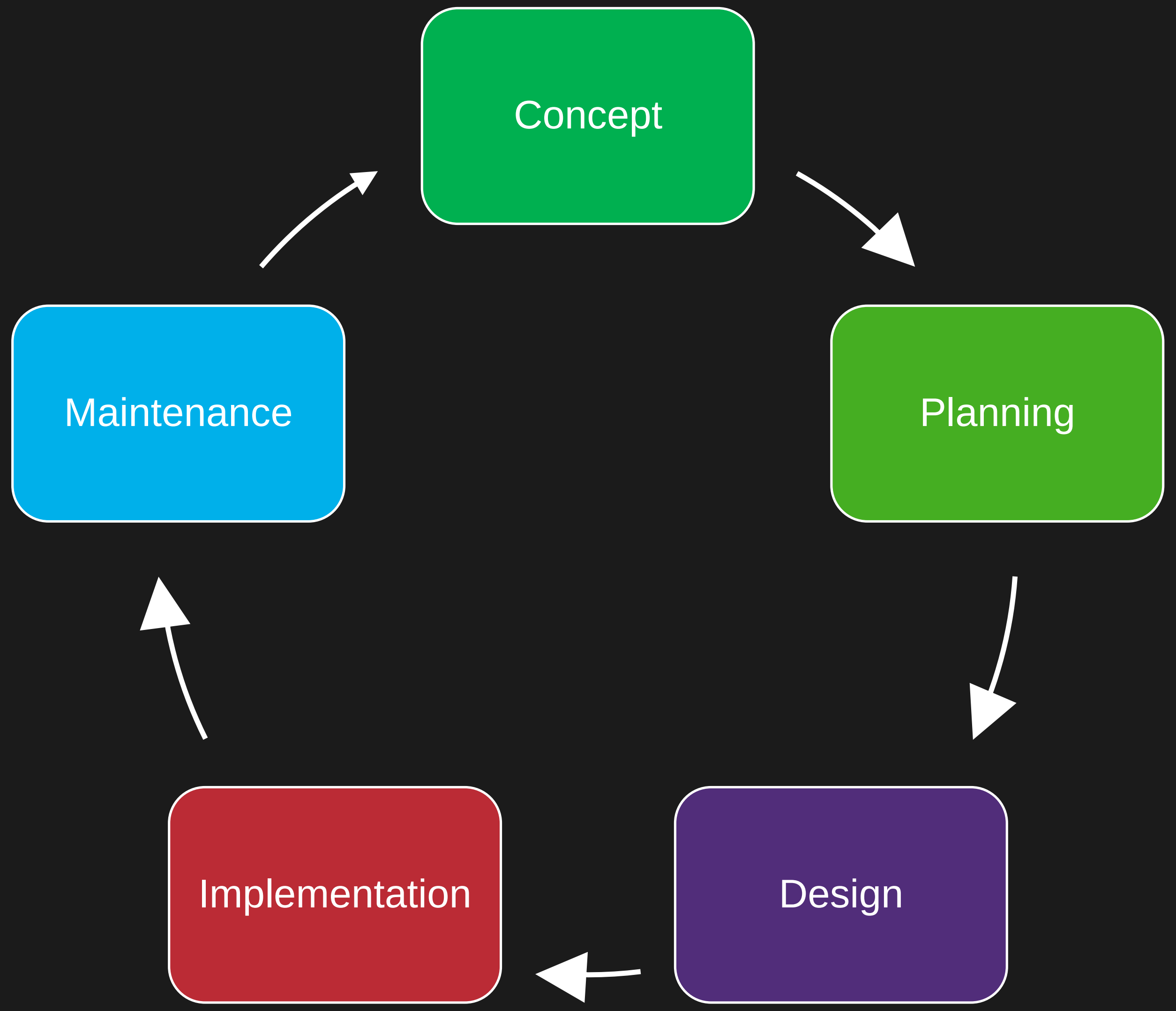


This Photo by Unknown Author is licensed under CC BY-NC-ND

WSJ

But WHY?!

Software Life Cycle



But WHY?!

- The code in a class is used by all instances and subclasses of that class.

Reuse is ***good***!

OOP: The Four Pillars

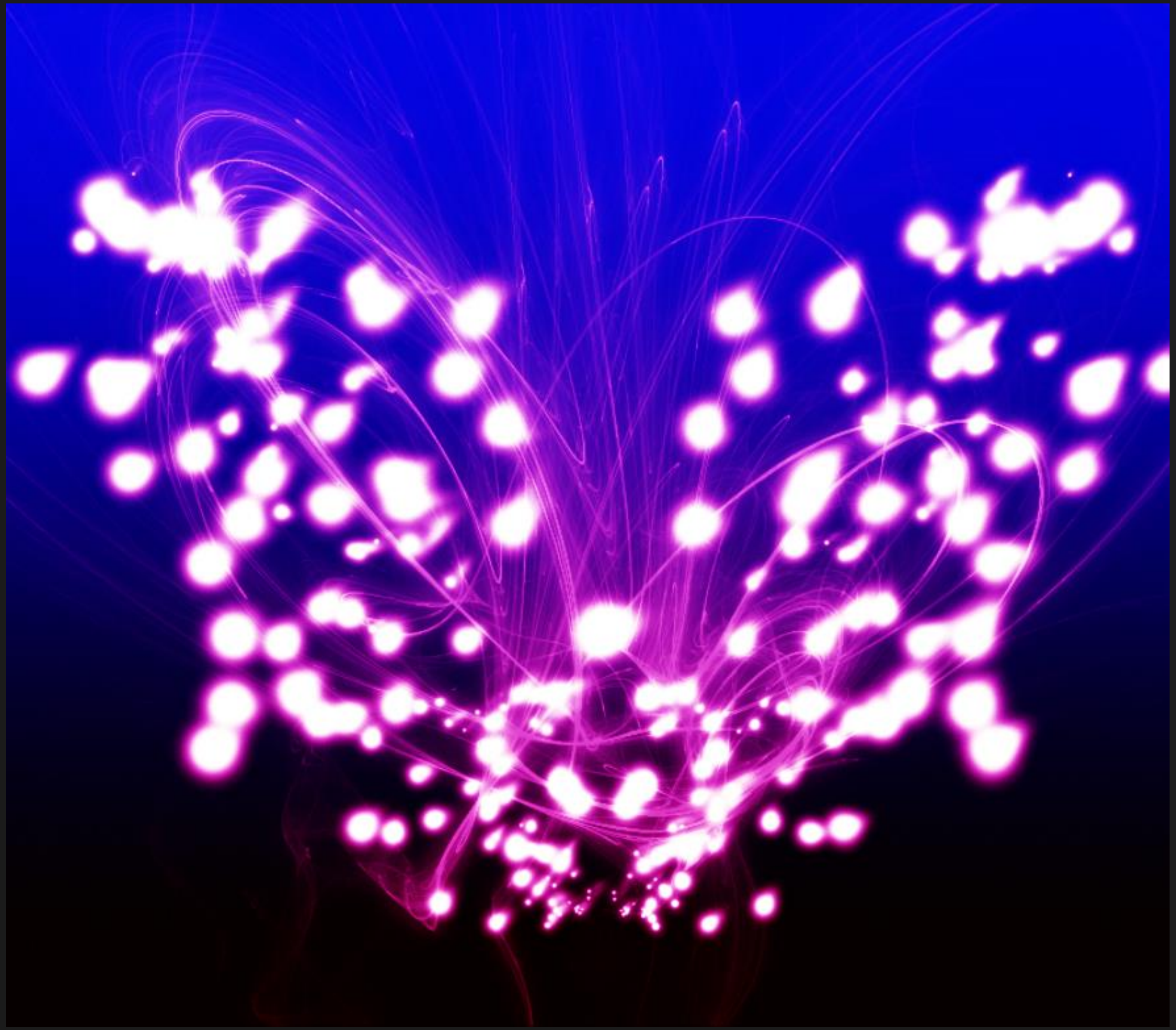
- The Four Pillars of OOP
  - **Abstraction**
  - **Encapsulation**
  - Polymorphism
  - Inheritance

OOP: Abstraction

- **Abstraction:** Ability to define objects that represent abstract entities that do work, change state, and interact with other entities

- It is about **grouping** certain behaviors (methods) and properties (data) of an object that **defines** the object abstractly

What do you see?



This Photo by Unknown Author is licensed under CC BY-NC-ND

OOP: Encapsulation

What’s inside? IDK

- **Encapsulation**: the grouping of data with the behaviors (methods) that do something with them
- **HIDES:**
  the data
  how it works
- The internal workings need to be hidden from the user of the class. **WHY**?
  - To protect the instances from being modified in a way that would make it invalid

Class Library

Create an console app + class library

1. Create a C# Console App
   - Choose .NET Core or .NET Framework
2. Add a class library to the solution
   - Make sure the kind of class library matches the console app. .NET Core for both or .NET Framework for both.

LINKS

[Creating C# Console App]

[Add another project]

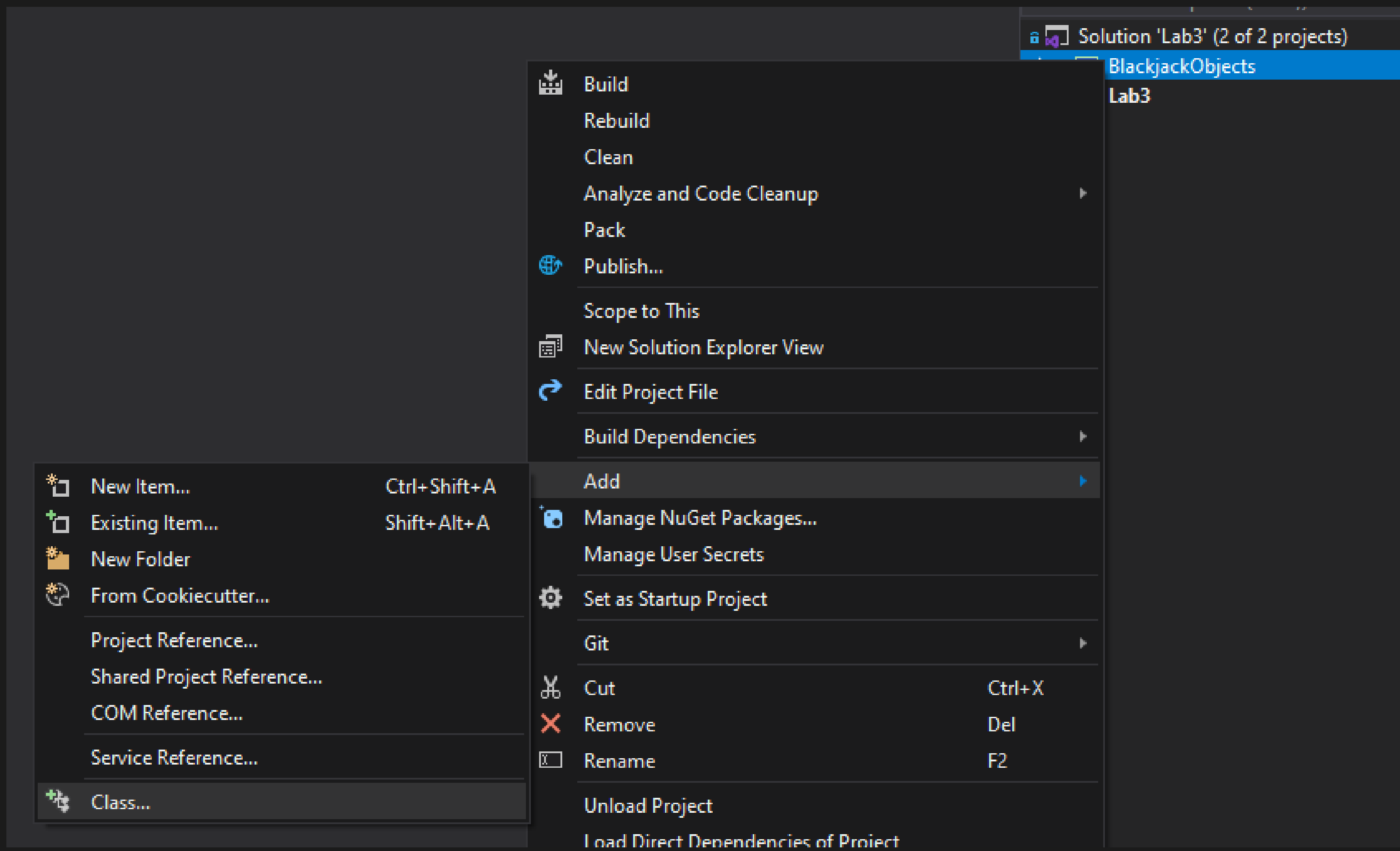
VIDEOS

Create a Class

Create a C# Class

- You should not put everything in the Class1.cs file.
- Good programming: put each class in their own file

1. Right-click the class library name in the Solution Explorer.
2. Select “Add->Class…” from the context menu.
3. In the dialog, enter the name of the class.



Fields

C# Classes: Fields

- Fields are the **data** for your class.
- Normally, you want to **hide** your data from the users of your class. This keeps them from modifying your data without your knowledge. You would do this by making the field private or protected.
- Fields follow Camel Casing naming convention.
  Each word within a compound word is capitalized except for the first.
  Examples: userFirstName, userLastName, dateOfBirth
- In C#, it is common to prefix field names with an _. This helps the reader of the code know that the variable is a field and not a local variable or a parameter.
  Examples: _userFirstName, _userLastName

C# Classes: Fields

```
class Car
{

        //instance fields. You need an instance of the class to access
        private int _odometerReading = 0; //can initialize here or in the constructor
        protected bool _isOriginalOwner = false;



         //static fields. You access with the class name. EX: Car._numberBuilt
        private static int _numberBuilt;
```

Inventory class

1. Add a class called Inventory to the class library
2. In the class, add the following fields:
   - capacity (an int that holds the max # of items)
   - items (a List of strings to hold the items)
3. Remember to use camel casing and follow encapsulation

Example:

```
class Car
{
        private int _odometerReading = 0;
        protected bool _isOriginalOwner = false;
```

LINKS

Classes

VIDEOS

Access Modifiers

C# Classes: Access Modifiers

- Access Modifiers control who can access an item in the class
- public: means anyone can access it
- private: means only the class can access it
- protected: means only the class and derived classes can access it

Properties

C# Classes: Properties

- Properties are the gatekeepers to the data of your class.
  They control the access and modification of your data.

- Traditionally, this was handled with getter and setter methods. In C#, they are handled with properties.

- Properties follow the [Pascal]() naming convention.
  the first letter of each word in a compound word is capitalized.
  Example: FirstName

C# Classes: Properties

- Properties have a get and a set. You can have both or just one but you have to have one of them.
- The get and set can have different access modifiers.

C# Classes: Properties

```
class Car
{
    //this is a full property with a backing field
    private int _odometerReading = 0;

    public int Odometer
    {
        get { return _odometerReading; }
        set {
            if (value >= 0) //value is the parameter to the setter
                _odometerReading = value;
        }
    }
```

C# Classes: Properties

```
class Car
{
    //this is an auto-property. The compiler will provide a backing field for the property
    public ConsoleColor Color { get; set; }

    //this property has a private setter but a public getter
    public int Year { get; private set; }
```

C# Classes: Properties

```
Car batmobile = new Car();

batmobile.Odomoter = 1000000; //this will call the setter of the property

int reading = batmobile.Odometer; //this will call the getter of the property
```

Properties Challenge

1. In Inventory, add the following properties:
   - Capacity: wraps the capacity field of Inventory.
     - Don’t let capacity go negative (check it in the setter)
   - Count: simply return the count of the list field. No setter is needed.
   - Items: return the list of items. Make the setter private.
2. Remember to use Pascal casing

Example:

```
private int _odometerReading = 0;
public int Odometer
{
        get { return _odometerReading; }
        set {
                if (value >= 0)
                    _odometerReading = value;
        }
}
```

LINKS

Classes

VIDEOS

Constructors

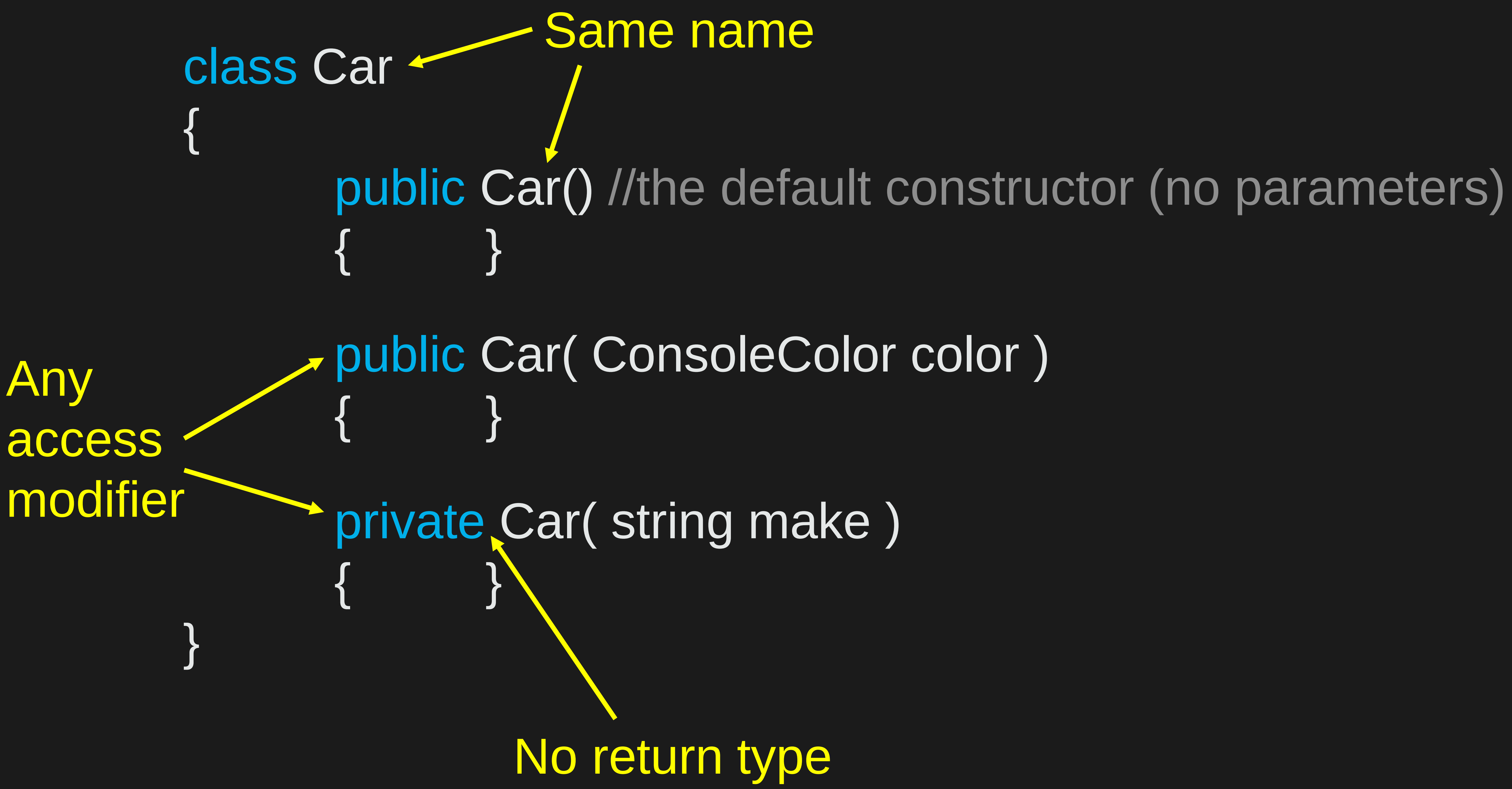
C# Classes: Constructor

- Constructors are special methods that initialize your objects.
- Most objects need to be set up by the constructor before other code starts to use them.
- When you see code like “ = new List<int>();” you are calling a constructor.

C# Classes: Constructor

- Constructors rules:
  - Cannot have a return value specified
  - **MUST** have the same name as the class (casing is important)
  - Can have as many as you want.
  - They can be public, private or protected.
  - A default constructor is provided by the compiler until you create a different constructor or your own default constructor.

C# Classes: Constructor



```
class Car
{
        public Car() //the default constructor (no parameters)
        {       }

        public Car( ConsoleColor color )
        {       }

        private Car( string make )
        {       }
}
```

Same name

Any access modifier

No return type

C# Classes: Constructor

The parameters passed to a constructor are used to initialize the class members (properties and fields).

```
class Car
{
        public ConsoleColor ExteriorColor {get;set;}

        public Car( ConsoleColor color )
        {
            ExteriorColor = color;
        }
}
```

Constructor Challenge

1. In Inventory, add a constructor that initializes capacity and items (meaning you need parameters)
   - Make sure you clone the list parameter

Example:

```
public Car( ConsoleColor color )
{
  ExteriorColor = color;
}
```

LINKS

Classes

VIDEOS

Methods

C# Classes: Methods

- Methods are the behaviors of your class (what it can do).
- Methods follow the Pascal naming convention.
  the first letter of each word in a compound word is capitalized.
  Example: UpdateScore()

C# Classes: Methods

```
class Car
{

        //instance methods. You need an instance of the class to access
        public void Honk() {  }



         //static methods. You access with the class name. EX: Car.CreateCar()
        public static Car CreateCar() { return new Car(); }
```

Method Challenge

1. In Inventory, add a method called AddItem that takes a string parameter for the item to add.
   - IF the count is less than capacity, add the item to the list.
     IF NOT, then throw an exception.

   EXAMPLE:

   - throw new Exception(“Danger!”);

LINKS

Classes

VIDEOS

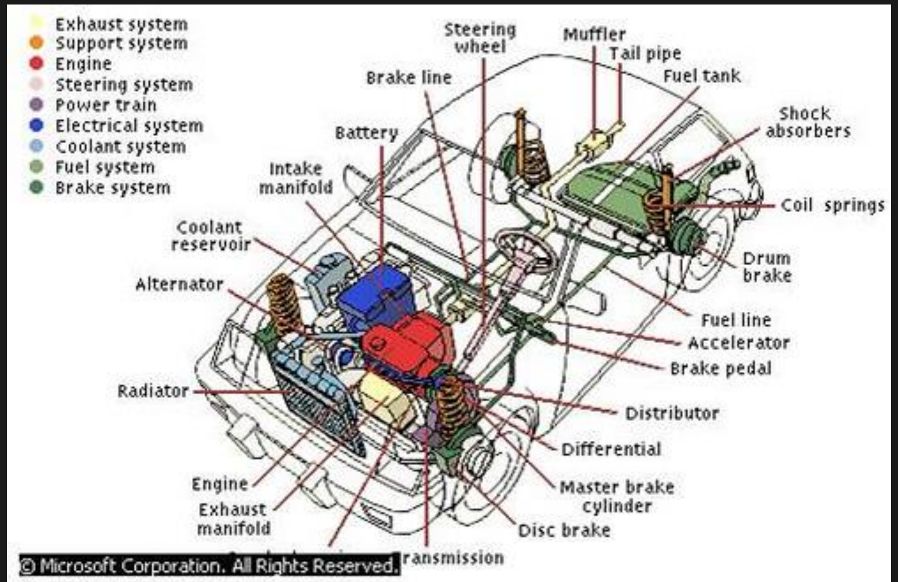
Instances

How-To Create an Instance

C# Classes: Instances

- Instances are 1 creation using a class.
  Example: you are an instance of a human.
- To create an instance, you use the new keyword.
  Example: Car beast = new Car(ConsoleColor.Yellow);
- Each instance of the class is separate from the others.
  Each instance has its own set of fields and properties.

OOP: Objects



This Photo by Unknown Author is licensed under CC BY-SA-NC

- Objects are represented with **classes** and **instances**.

- A Class ***defines*** the data and behavior.

- An Instance ***represents*** 1 creation of a class. When a car exits the assembly line at the Tesla factory , you have an instance of a car.

C# Classes: Instances

- To create an instance of the Car class:

```
Car myFirst = new Car( ConsoleColor.Blue );
```

- NOTE: you can NOT create an instance of a static class or an interface.

Instance Challenge

1. In Main, create an instance of Inventory and add several items to it.

LINKS

Classes

VIDEOS

Class Challenges

1. In the class library, create an enum called WeaponRarity
   - Add the following to the enum: Common, Uncommon, Rare, Legendary
2. Create a class called FantasyWeapon
   - Add the following properties:
     - Rarity (use the WeaponRarity as the ***type***)
     - Level (an int)
     - MaxDamage (an int)
     - Cost (an int)
   - Add the following method:
     - DoDamage. Should return an int. It should calculate the damage by multiplying max damage by a randomly picking number between 0-1.
   - Add a constructor to FantasyWeapon that takes in parameters for rarity, level, maxDamage, cost.
     - Be sure to set the properties with the parameters of the constructor.
3. In Main, create an instance of FantasyWeapon.
   - Call DoDamage on the instance and print the damage.

INTERMEDIATE LEVEL:

Interfaces

For extra learning

Classes & Interfaces

- an interface is the *public* data and methods
    - It is how the *public* can use (or interface) with your class.

- Real world example: the steering wheel, gas pedal and brake are the public interface for how a driver interacts with the car.

C# Interfaces

- You can create special types in C# called an interface.
  - It allows you to set the guidelines for what the public interface of a class should be.

- The interface only defines what the signatures look like – they are not meant to provide the code for those methods.

Interface Rules

- Because there is no code in the interface, you cannot create an instance of an interface.
- Everything defined in an interface is public. You cannot specify an access modifier.
- Classes that implement an interface must provide code for all parts of the interface.

Programming to an Interface

- By implementing the interface, the class is *guaranteeing* that it will implement specific methods.

- Other code can just use the interface as the type instead of the class and not care about what class is being used. This is called programming to an interface.

EXAMPLE:

```
IPlayer player1 = GetPlayer(); returns some class instance that implements IPlayer
player1.Health = 100;
```

Define an Interface

```
public interface IPlayer  //names usually start with I
{
        int Health {get; set;}  //properties (no code)
        void UpdatePosition(); //methods (no code)
}
```

No access modifier (they are public)

Implement an Interface

- Everything in the interface must be implemented as public in the class.

```
public class Player : IPlayer
{
    public int Health {get; set;}
    public void UpdatePosition()
    {
            //add code here specific to the class
    }
}
```

Enum Challenge

1. In the class library, create an enum called WeaponRarity
2. Add the following to the enum: Common, Uncommon, Rare, Legendary

LINKS

Enums

VIDEOS

Interface Challenge

1. Create an interface called IWeapon
2. Add the following properties:
   - Rarity (use the WeaponRarity as the ***type***)
   - Level (an int)
   - MaxDamage (an int)
   - Cost (an int)
3. Add the following method:
   - DoDamage. Should return an int

LINKS

Interface

VIDEOS

Implement Challenge

1. Create a class called FantasyWeapon
2. Implement IWeapon
   - You’ll need to implement the properties and method.
   - Make DoDamage calculate the damage by multiplying max damage by a randomly picking number between 0-1.
3. Add a constructor to FantasyWeapon that takes in parameters for IWeapon (rarity, level, maxDamage, cost).
   - Be sure to set the properties with the parameters of the constructor.

LINKS

Classes

VIDEOS

Instance Challenge

- In Main, create an instance of FantasyWeapon.
- Call DoDamage on the instance and print the damage.

LINKS

Create an instance

Calling methods

VIDEOS