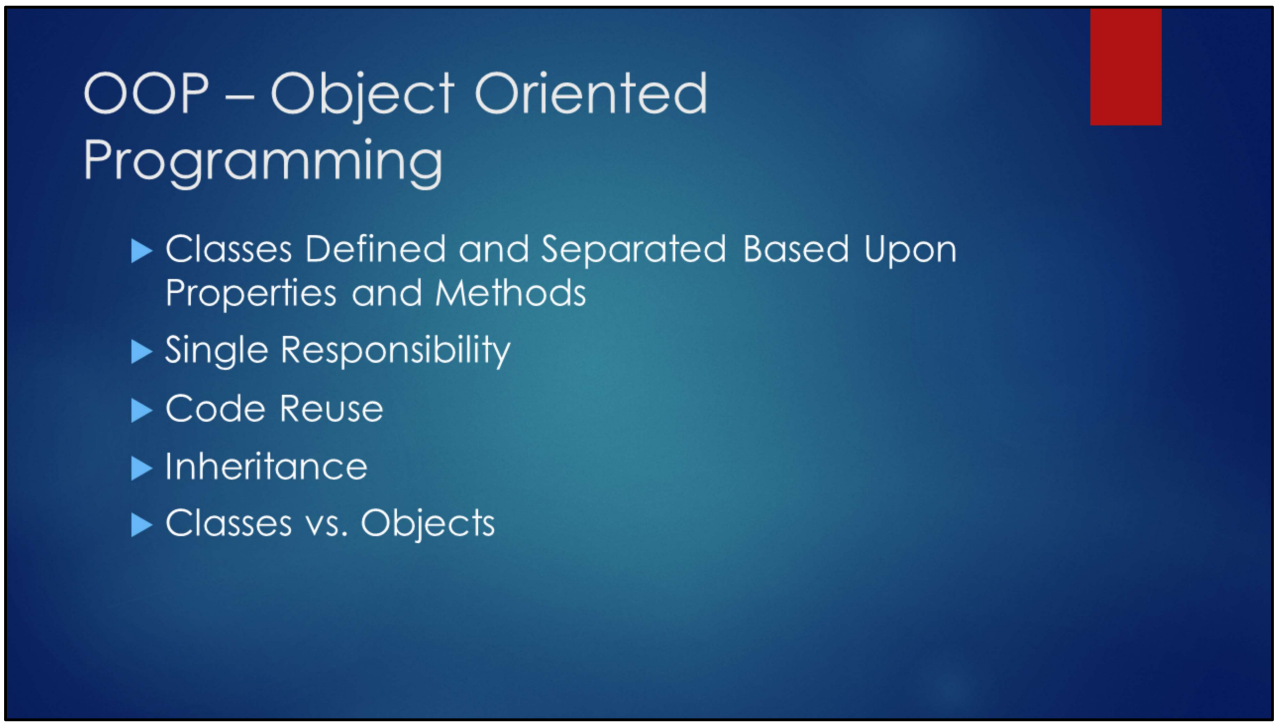
Logols Learning

WEEKEND WEB DEVELOPMENT BOOT CAMP

TRAINING: OBJECT ORIENTED PROGRAMMING



- Object Oriented Programming is a way of programming in which code is separated into objects or classes.
- In this way it gives the programmer a way to have code represent something and to have single responsibility.
    - Single responsibility means that the class is only responsible for one thing and that it is only being used for one use case (or group of people that would use it).
- This usually makes for more code reuse instead of just writing code throughout the program when you need it.
- There is also the ability to inherit properties and methods from a base class.
- So, for example you could have a shape base class with a height property and width property.
- A square which is a particular type of shape would need these same properties and so could inherit from the shape class to use these same properties.
- In .Net there is a difference between a class and an object.
- The class is the definition.
- The object is what is created each time a class is instantiated.
- There are alternatives to object oriented programming such as procedural and functional programming.
    - All of these paradigms are now used together.

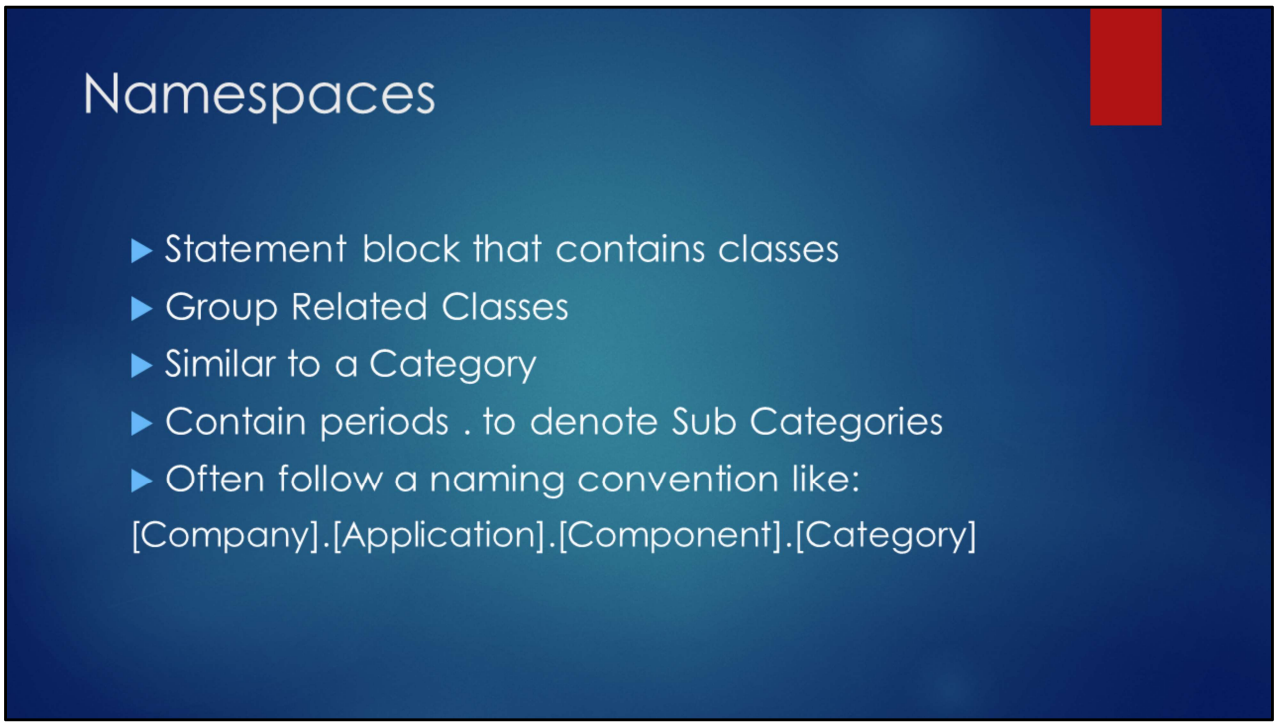
- Classes are statement blocks that contain methods and properties and fields.
- The methods act as the actions or behaviors of the object.
- The properties and fields are used to store data for the object.
- Classes contain grouped together data and actions that mimic something similar to objects in the real world (tangible or intangible).
- Classes should have a single responsibility, meaning they should not do too much and should represent one use case from one set of users.
- The class is the definition, the object is the instance of that class.

Class Syntax

```
[access modifier] class [name] : [base class],
[interface1], [interface2]
{
    Statements…
}

public class Car : Automobile, IPositionWriter
{
  Statements…
}
```

- Classes have an access modifier which we will discuss later.
- They can inherit from one base class.
- They can implement multiple interfaces.



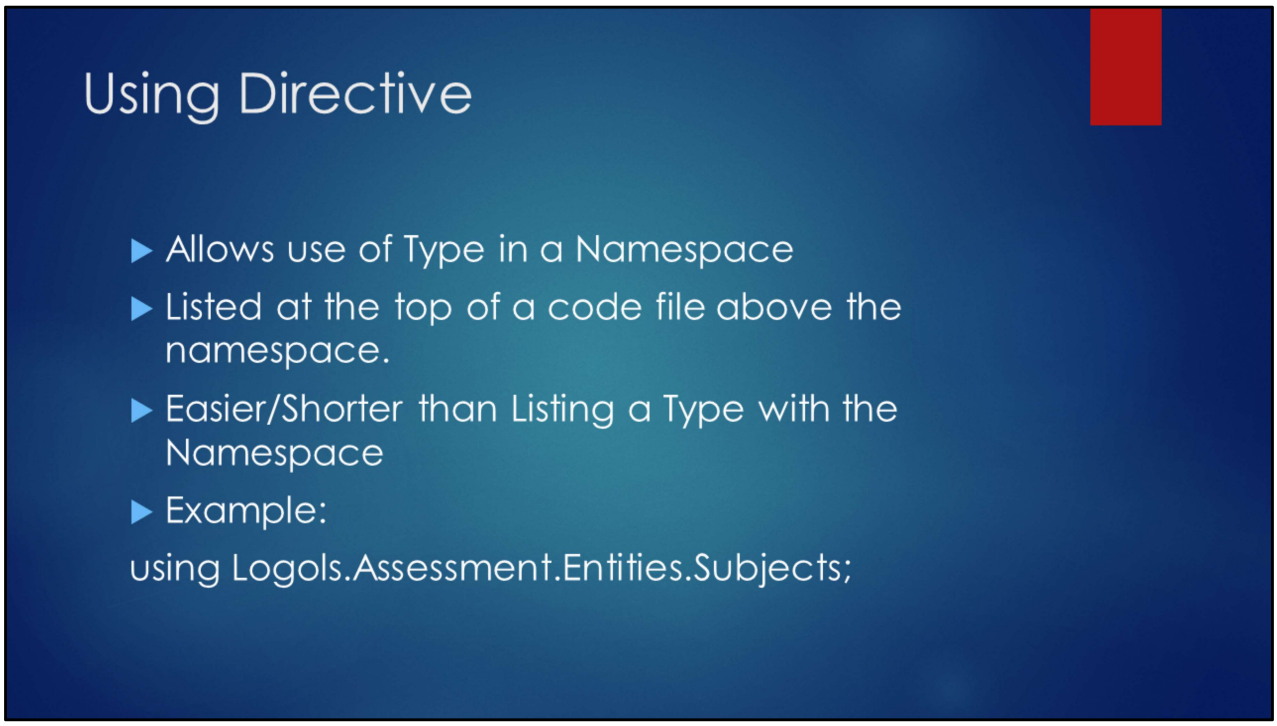
- Namespaces are statement blocks that contain classes.
- Namespaces group related classes together similar to a category.
- Periods . are used to denote sub categories.
- A naming convention is often used like:
  [Company].[Application].[Component].[Category]

Namespace Syntax

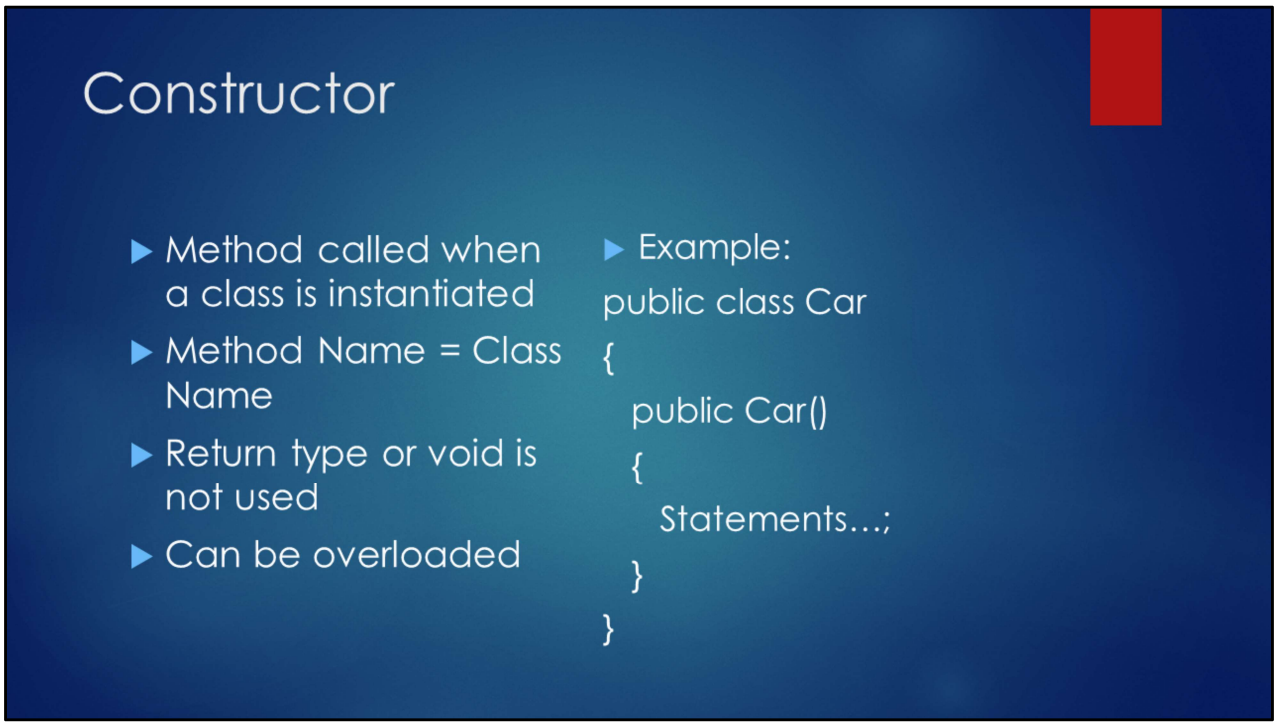
```
namespace [name]
{
  Statements…
}

namespace Logols.Assessment.Entities.Subjects
{
  Statements…
}
```

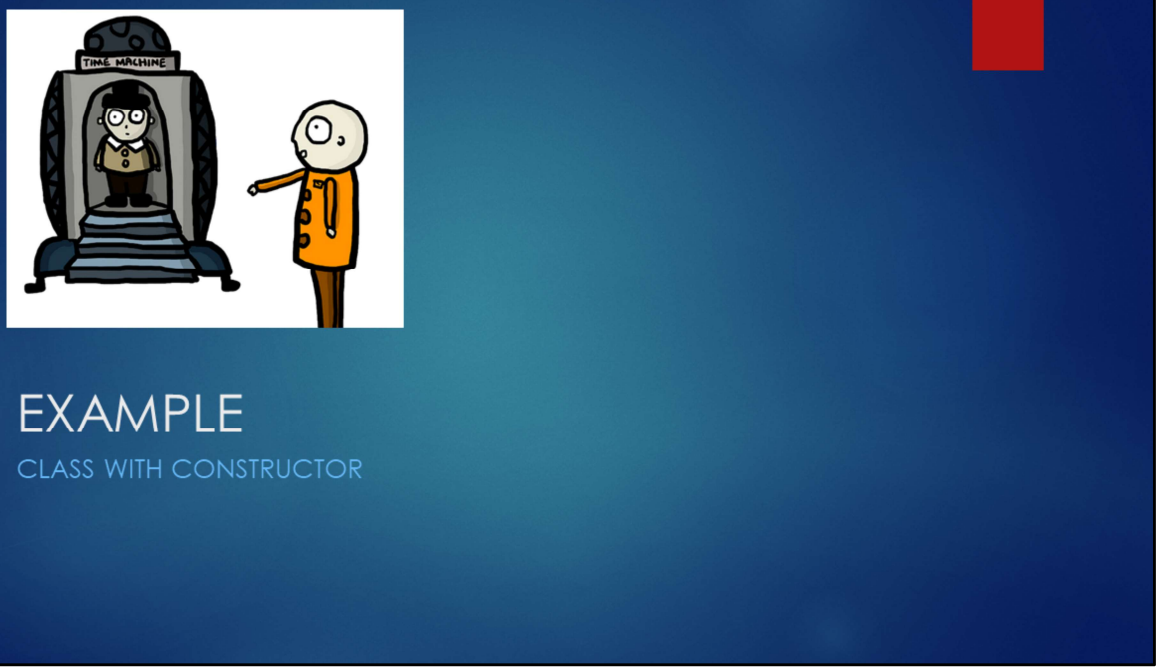
- The namespace is straightforward.
- The namespace keyword is specified followed by the name.



- A using directive allows the use of a type within a namespace in a class with a different namespace.
- The using statements are listed at the top of the code file.
- You could list types by using the full namespace and type name, but the using directive allows for a shorthand and looks cleaner.



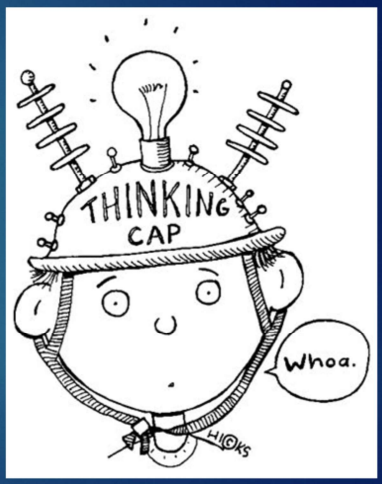
- A constructor is a method that is called when a class is instantiated.
- The name has to match the name of the class.
- No return type or void is specified for this method.
- The constructor can be overloaded like any other method, which requires different parameters with different types.



Let’s go through some examples

ASSESSMENT

CLASS WITH CONSTRUCTOR



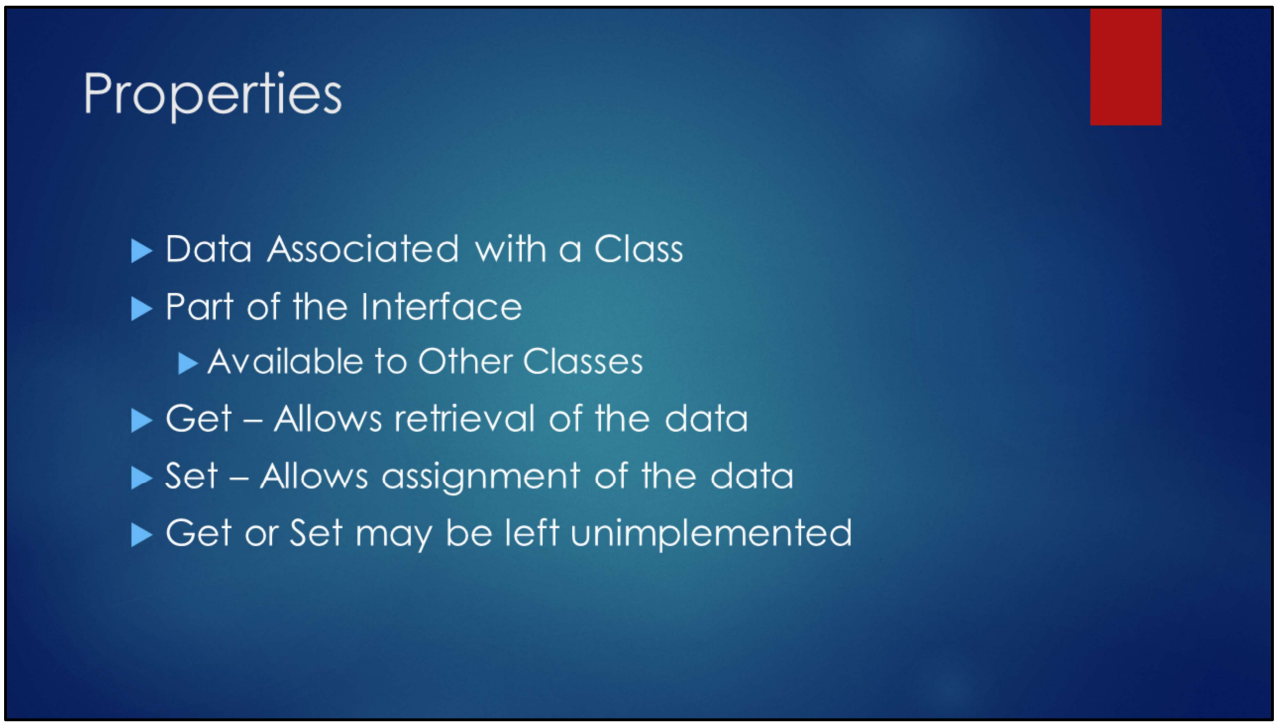
- What are statement blocks that contain methods and properties?
- Write on the board a public class with the name Car.
- Write on the board a namespace named Vehicle.
- Write on the board a using directive to bring in the Vehicle namespace.
- Write on the board a public class with the name Car with a constructor.

Assignment

- We have been surrounded by zombies and need to change camps. We will move by foot. The zombies will likely follow.
- A simulator has been requested to see how long it will take the zombies to catch up to us after we move, to decide the best location.

Assignment

- Create a new console project named Zombie.Simulator.
- Create a Person Class within the namespace Zombie.Simulator.
- Create a constructor that writes to the console: "A new person has been created."
- From the Main method, instantiate the Person class.



- Properties represent data associated with the class or object.
- Properties are available outside of the class and are considered part of the interface of the class.
- There is a get part of the property which allows retrieval of data.
- Also, there is a set part of the property which allows assignment of data.
- Only one part (the get or the set) is required to be implemented.
    - If get is not implemented, then the user of the class would not be able to retrieve data.
    - If set is not implemented, then the user of the class would not be able to assign data.

Property Syntax

General Syntax

```
[access modifier] [type] [name]
{
  get
  {
    Statements...
  }
  set
  {
    Statements...
  }
}
```

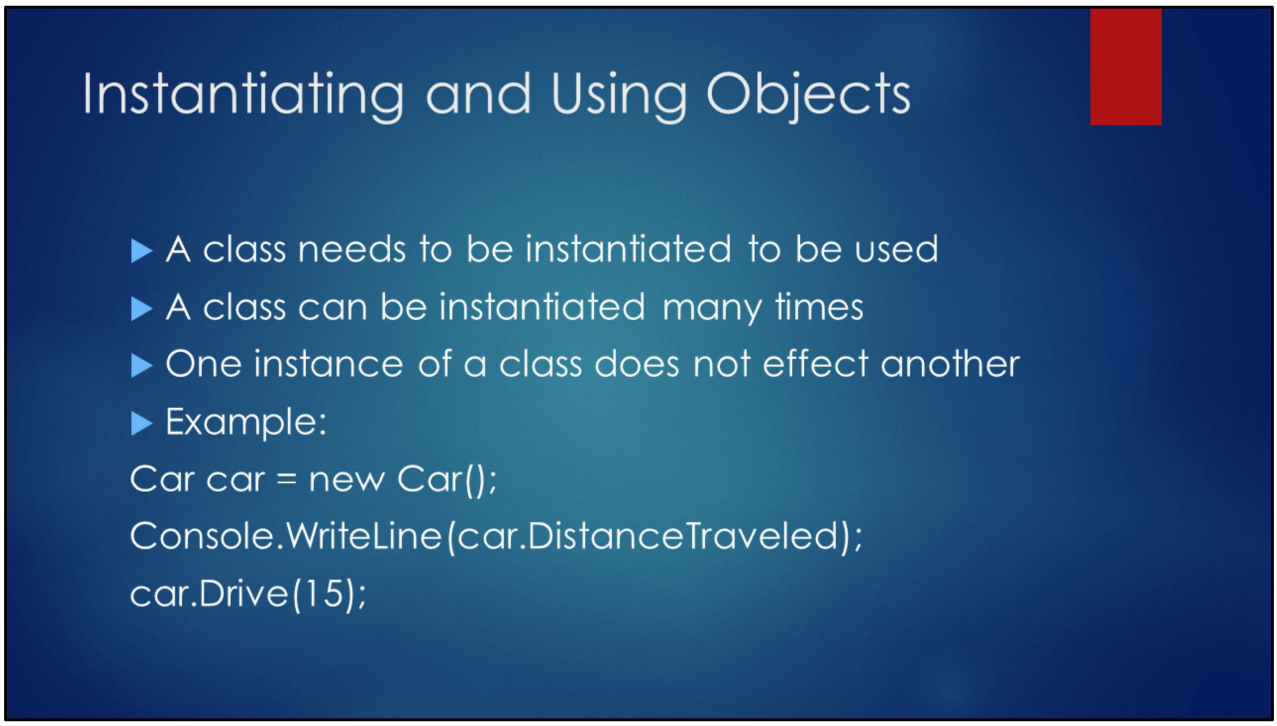
Fully Implemented Example

```
public int Count
{
  get
  {
    return _count;
  }
  set
  {
    _count = value;
  }
}
```

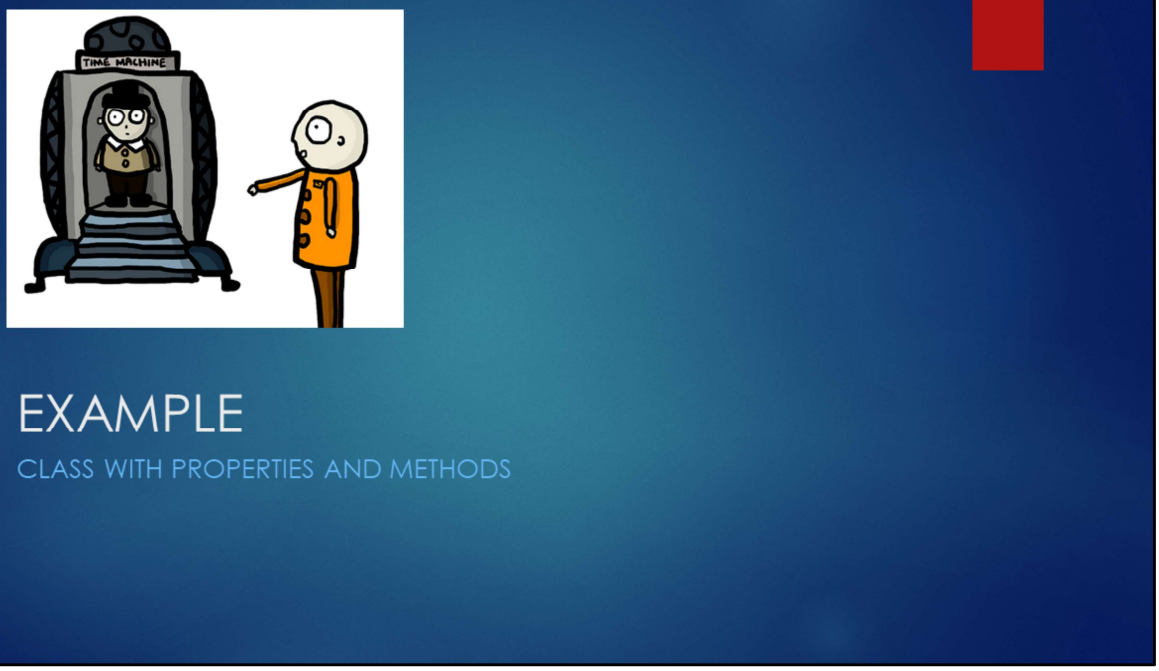
Auto Implemented Example

```
public int Count { get; set; }
```

- Properties have a shorthand syntax called auto implemented properties.
- In these methods a variable is created that underlies the property.
    - This variable is assigned in the set and retrieved from the get.



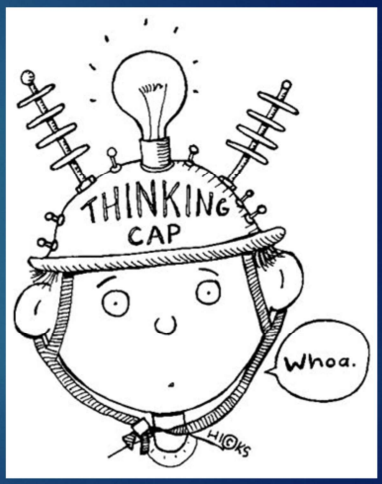
- A class needs to be instantiated to be used unless it is static.
- A class can be instantiated as many times as needed.
- Changing the data or calling a method in one instance of a class does not effect the data in another instance of the class.
    - Be careful though, if variables are pointing to the same instance of a class then changes will effect both variables.



Let’s go through some examples

ASSESSMENT

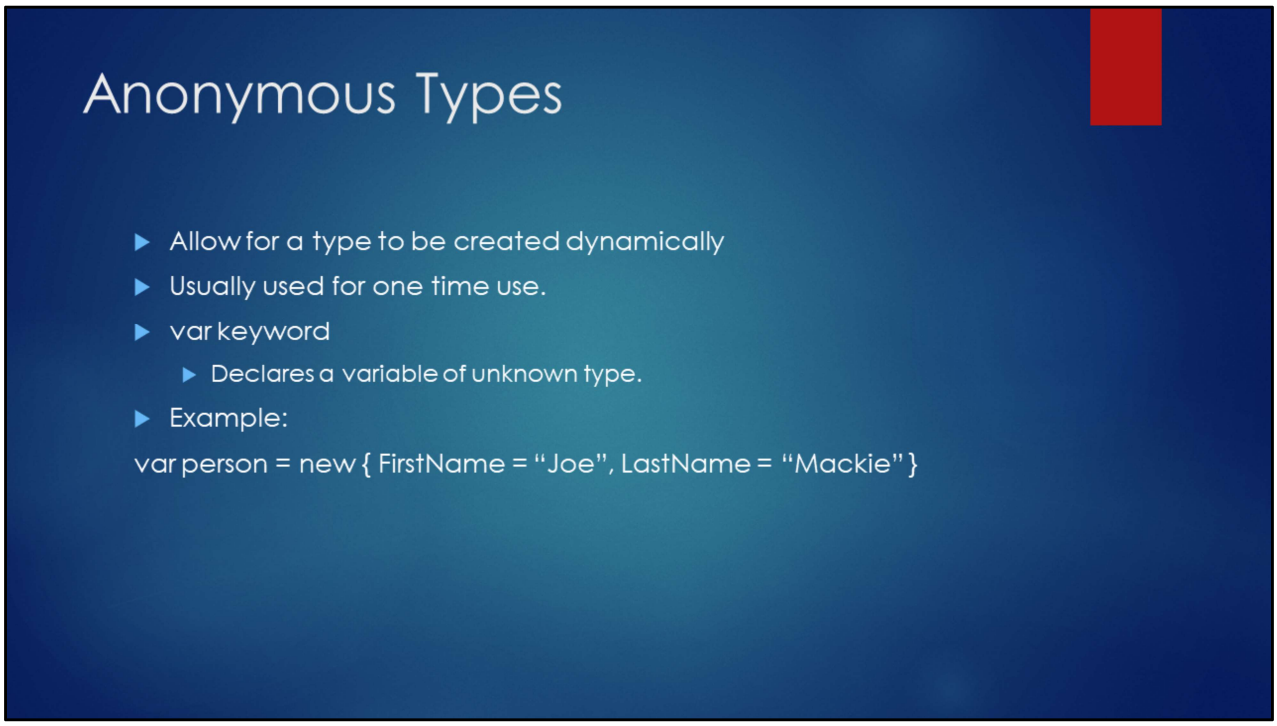
CLASS WITH PROPERTIES AND METHODS



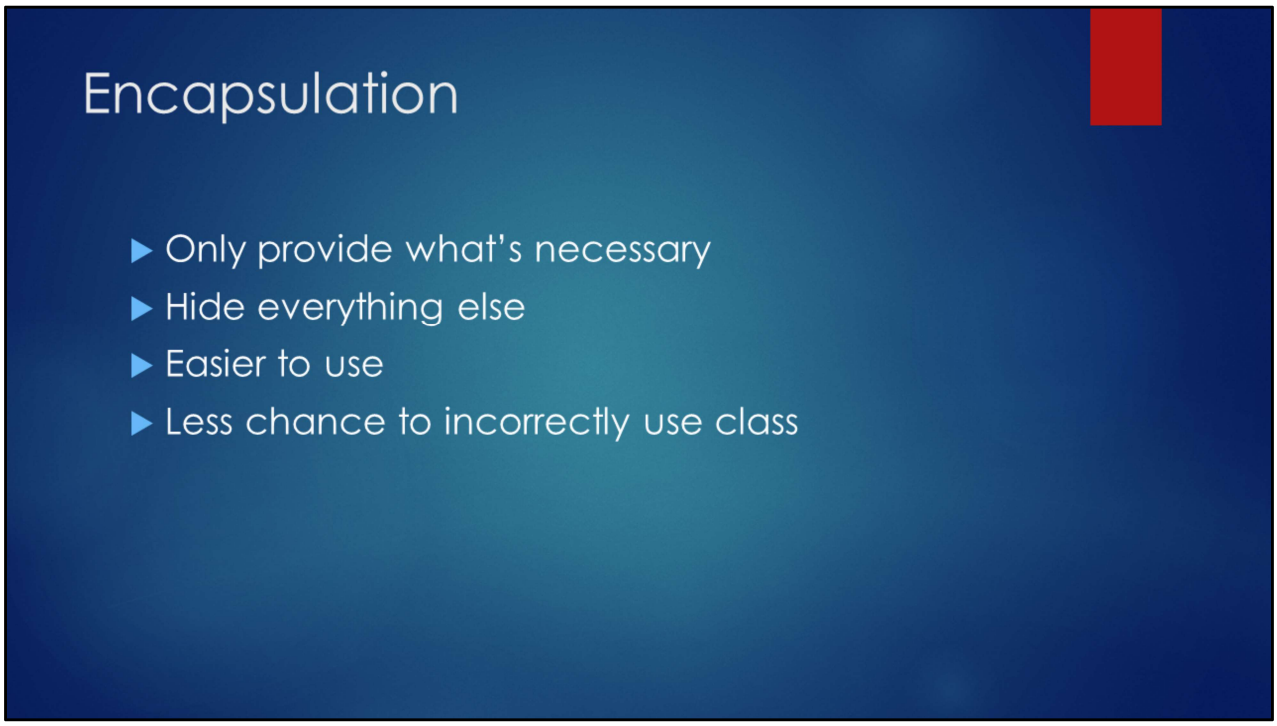
- Write on the board an auto implemented property named Miles with a type of int.
- Declare and instantiate a class named Car. Console the Miles property.

Assignment

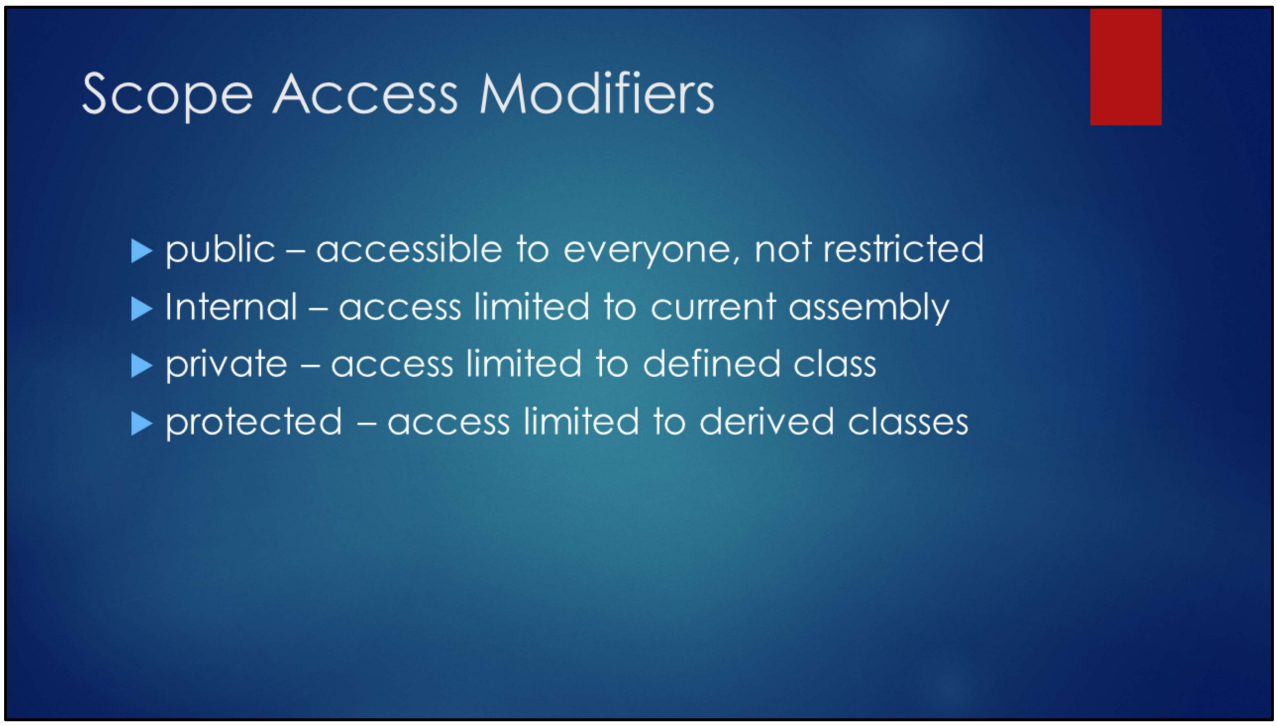
- Add to the simulator project.
- Use the general property syntax to create a new property named DistanceTraveled. Use an underlying field to store data.
- Practice calling and setting this property from the Main method.
- Modify this to an auto implemented property.
- Try instantiating multiple objects of the class.



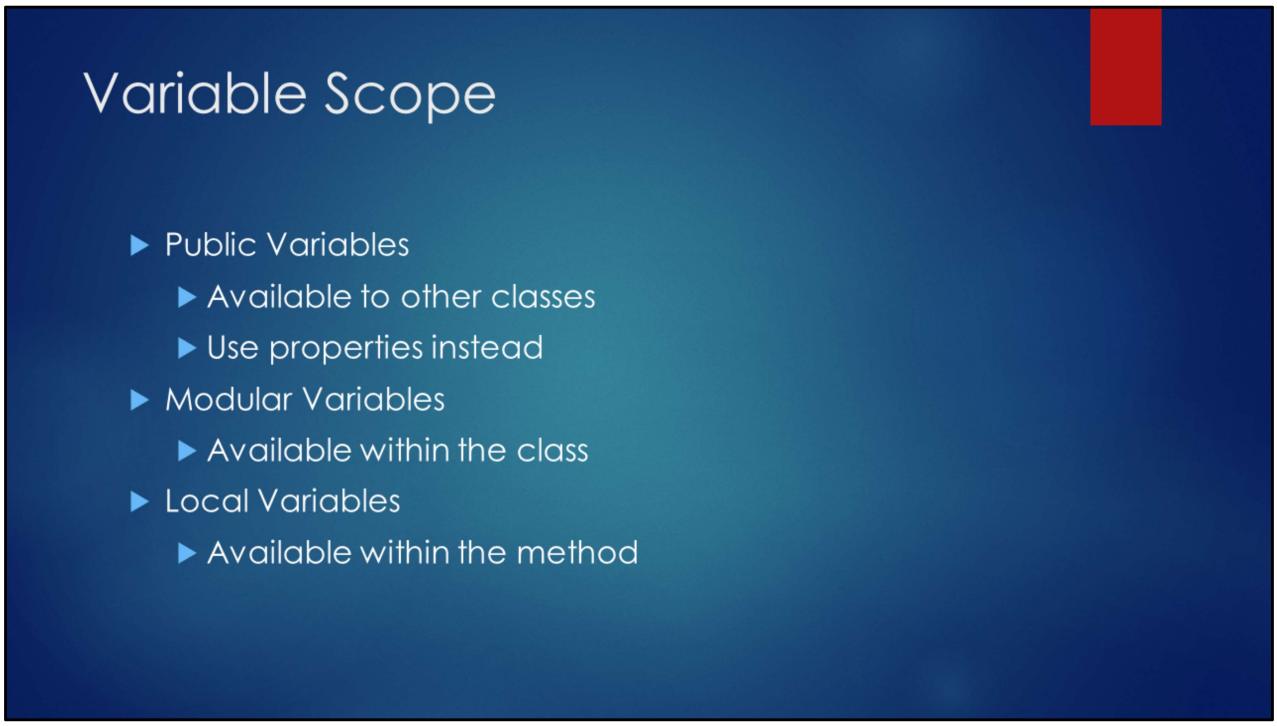
- Anonymous types allow for a type to be created dynamically.
- This would usually be for one time use, since you would want to define it as a class if you were going to be using it over and over again.
- The var keyword can be used to declare a variable and get it's type based upon which instance of an object is assigned to it.
    - This is ideal for anonymous types.



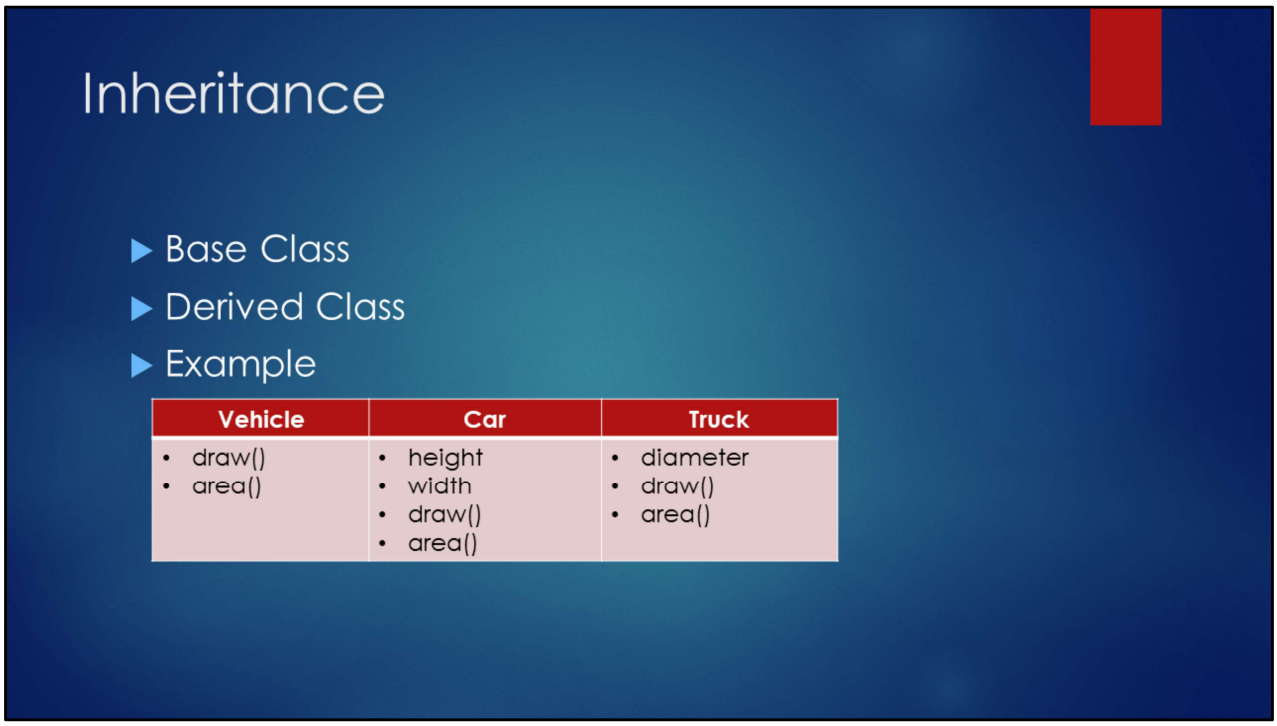
- Encapsulation is the objective to only provide what is necessary to the user of a class.
- Any other details of the class should be hidden to the user of a class.
- This makes the class easier to use, because there is less information to take in.
- There is also less chance to use the class incorrectly.
    - Meaning if there are dependencies on how methods should be called then the class itself can figure that out and not allow methods to be called by users of the class directly.



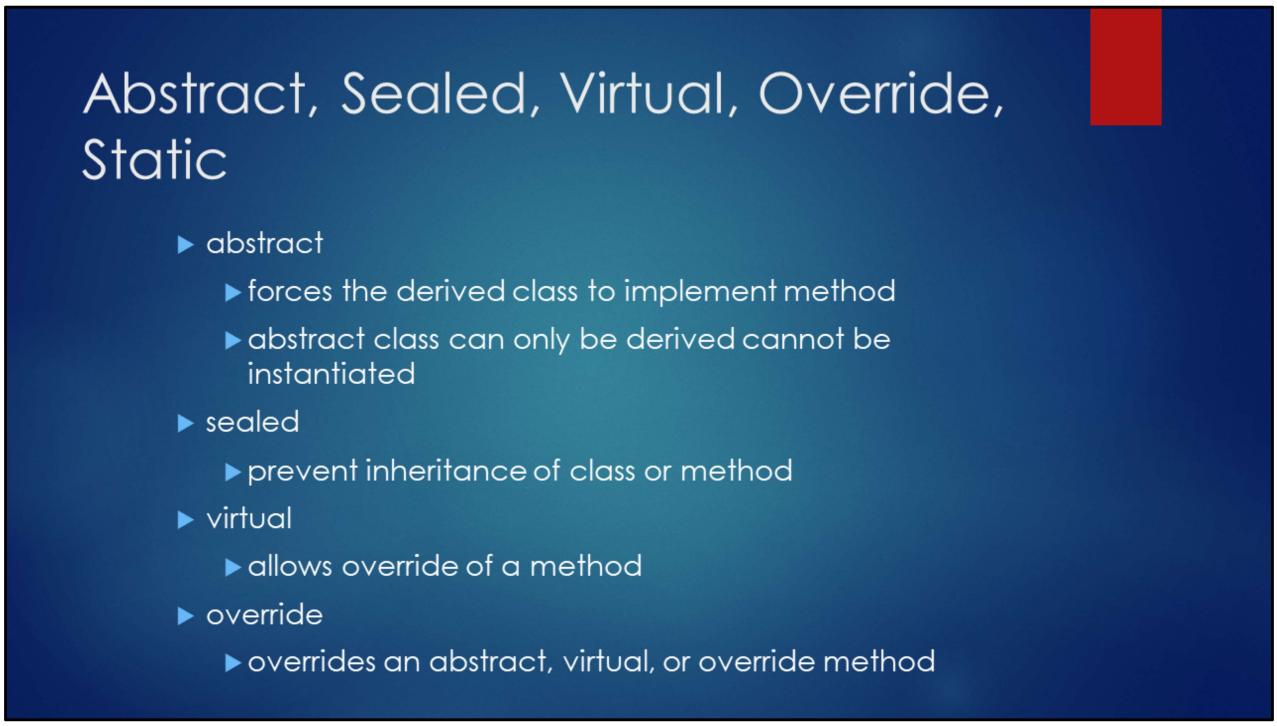
- Scope access modifiers specify who can access what code where
- These can be used on classes, methods, properties, or fields.
- If it is defined as public it means that code can be accessed by everyone from anywhere.
- If it is defined as internal it means that code can be accessed by other classes within the same assembly (project).
- If it is defined as private it means that code can be access only within that class.
- If it is defined as protected it means that code can be accessed only by derived classes.



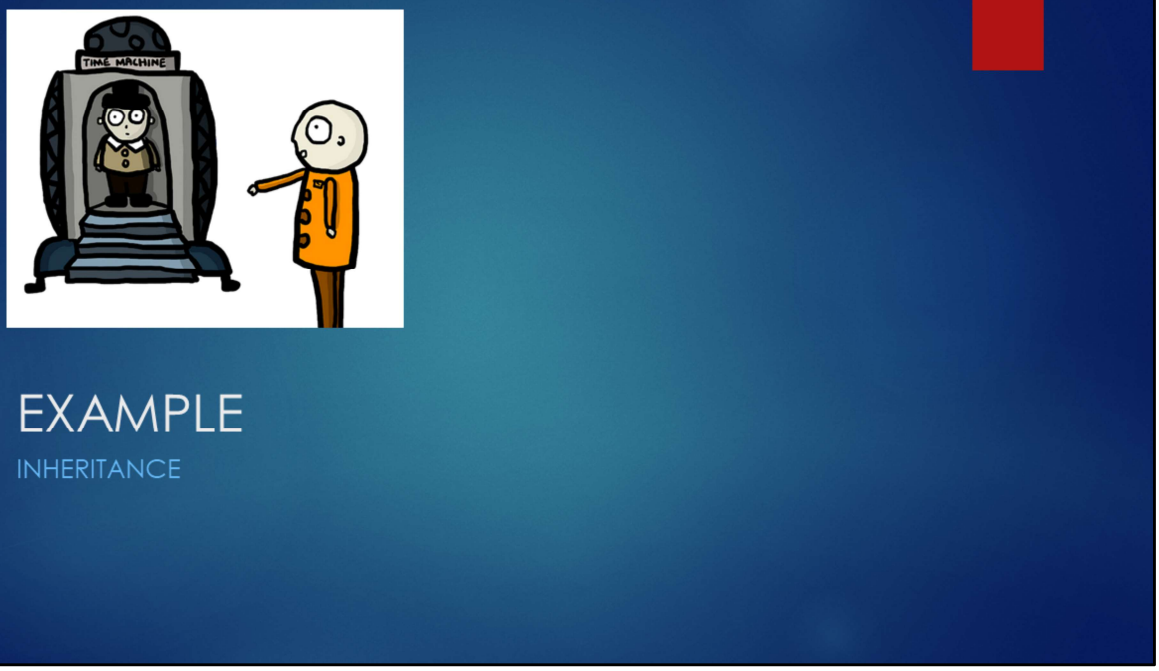
- Variables have scope depending upon where they are defined and the scope access attached to them.
- Those defined as public are available everywhere.
- Those defined as private within the class are available anywhere in the class. Also known as modular variables.
- Those defined in a method are local and can only be accessed within that method.



- Here's a more detailed example of inheritance.
- A derived class inherits from the base class
- This means that all properties, fields, and methods of the base class now belong to the derived class.
- In this shapes example there are a number of methods available in object.
- Since all types inherit from object these methods exist in our shape class.
- We add the draw and area methods to the shape class
- Then we create a rectangle and circle class
- Both of these can inherit from the shape class to gain the draw and area methods.
- Then we differentiate these because we will only use the diameter property for the circle, but we will add height and width to the rectangle class.



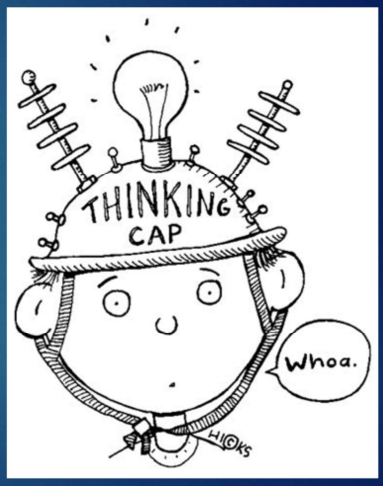
- A class or method can be specified as abstract.
- For a class this means that it can not be instantiated directly. The derived class must be instantiated instead.
- For a method it means that there is no implementation of the method, but the derived class needs to implement the method.
- A class or method can be specified as sealed.
- This means that the class cannot be inherited or the method cannot have an override.
- Virtual means that the method can have an override.
- Override means that the method is overriding what is in the base class. So, run the logic in the derived class instead of in the base class.



Let’s go through some examples

ASSESSMENT

INHERITANCE



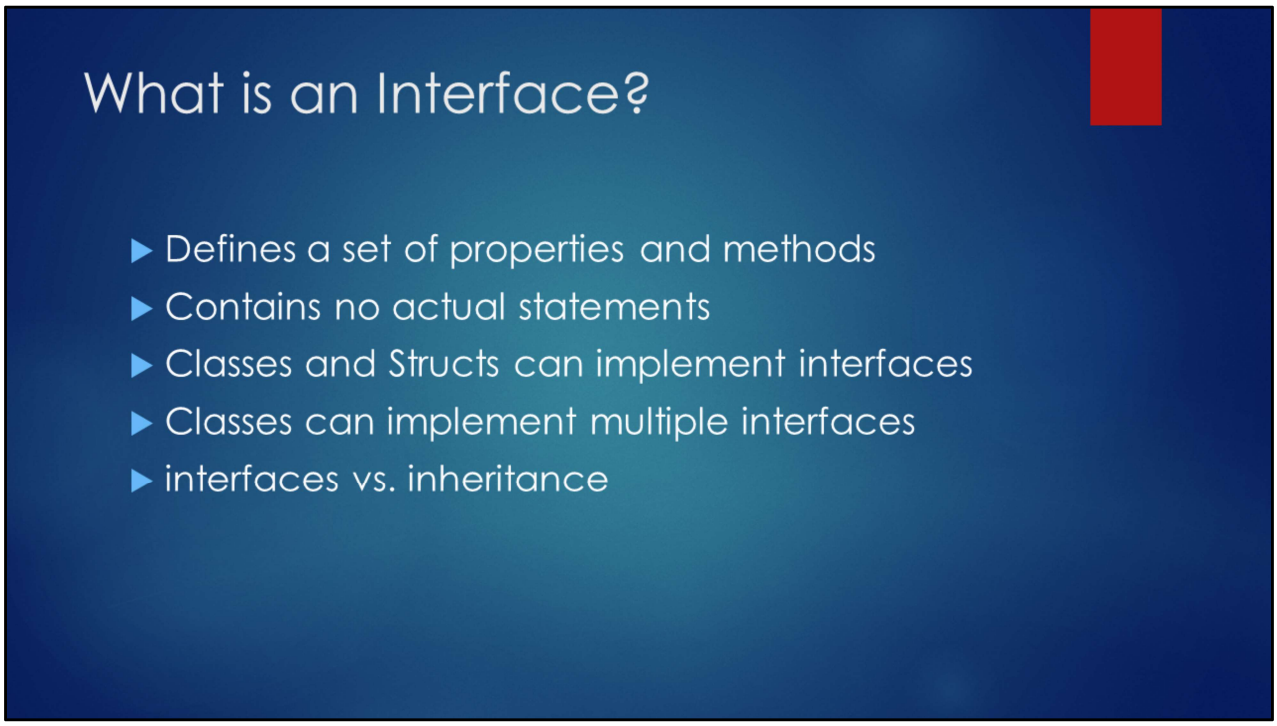
- Write on the board a declaration of a variable set to an anonymous type with properties miles and size.
- What are the 4 scope access modifiers?
- Write on the board a class named Car that inherits a Vehicle class.
- Write on the board a class named Car that cannot be derived.
- Write on the board a class name Car that has to be derived.
- Write on the board a method that overrides a base class method called Drive that returns a decimal and takes a parameter of decimal named miles.

Assignment

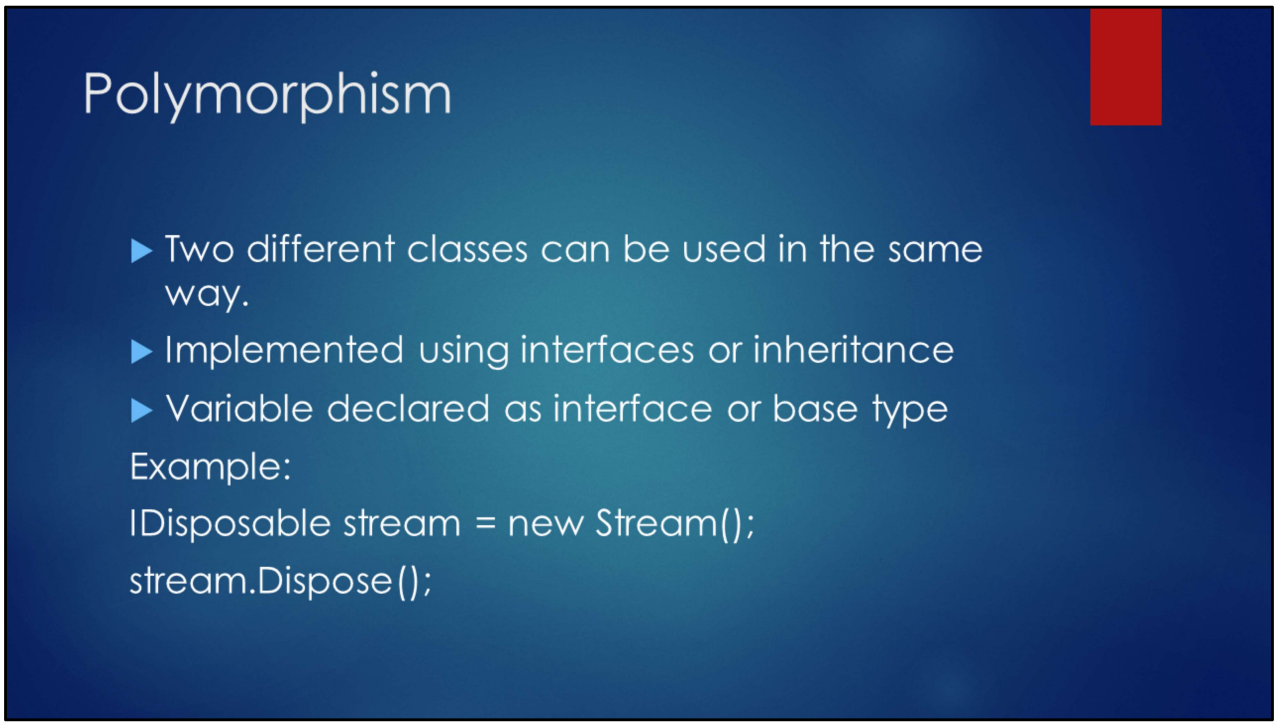
- Add to the simulator.
- Make the Person class abstract.
- Make a Human and Zombie class that both derive from the Person class. Make these sealed.
- Create a virtual method Walk in the Person class that takes a decimal parameter named minutes and sets DistanceTraveled based on a calculation.

Assignment

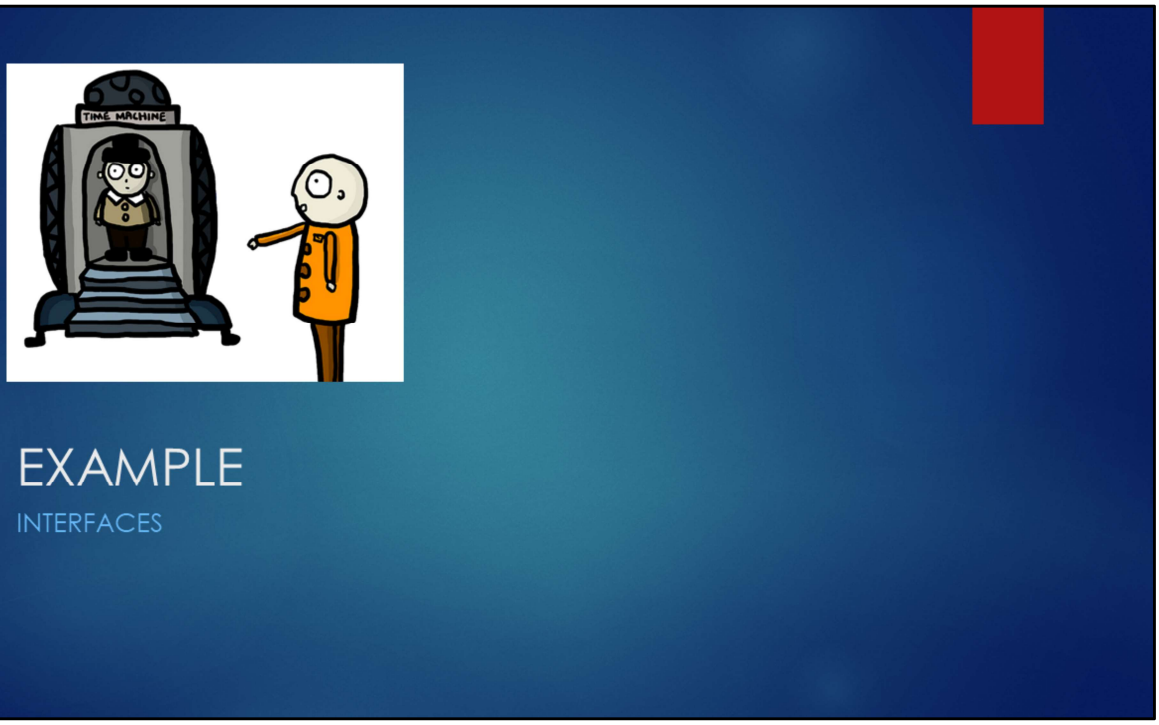
- Create an override method of Walk in the Zombie class to change the calculation to be slower.
- Create a method named Run in the Human class that takes a parameter named minutes. Calculate and set DistanceTraveled. This should be faster than the walk methods.
- Create instances of Zombie and Human. Call the walk and run methods and see how far they travel.



- An interface defines what needs to be implemented.
- It has no implementation itself.
- It contains properties and methods.
- Classes and structs can implement interfaces.
- Multiple interfaces can be implemented by one class.
- Implementing interfaces is not the same as inheriting as there is no logic. The logic needs to be implemented by the class.



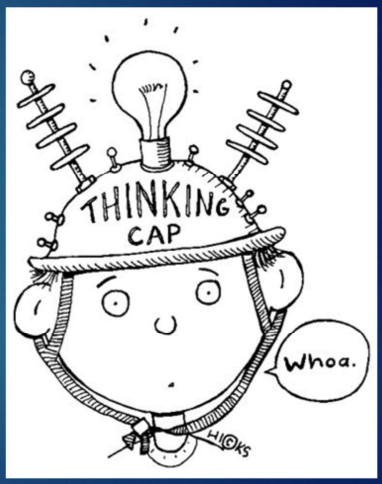
- Polymorphism essentially means that two different classes can be used in the same way, without caring about the actual implementation.
- This can be implemented by using interfaces or inheritance.
- The user can declare variables as of the interface or base types instead of the actual type class they are using.
- In this way the user can interact with classes only caring about what is available in the interface or base type and not caring about the differences of the actual class type.



Let’s go through some examples

ASSESSMENT

INTERFACES



- Write on the board a class named Car that implements an interface named IDrivable.
- When the user can interact with a base class or interface in a similar way without caring about the underlying implementation, it is called what?

Assignment

- Create an interface IPerson that defines a property DistanceTraveled and a Method Walk.
- Modify the Zombie and Human classes to implement from this interface instead of derive from the Person class.
- Instantiate multiple versions of each class, but set them equal to a variable defined as IPerson and add them to a List of IPerson
- Loop through each instance calling the walk method using polymorphism.

QUICK REVIEW

OBJECT ORIENTED PROGRAMMING

- Write on the board a public class with the name Car.
- Write on the board a public class with the name Car with a constructor.
- Write on the board an auto implemented property named Miles with a type of int.
- Declare and instantiate a class named Car. Console the Miles property.
- Write on the board a class named Car that inherits a Vehicle class.
- Write on the board a class named Car that cannot be derived.
- Write on the board a class name Car that has to be derived.
- Write on the board a method that overrides a base class method called Drive that returns a decimal and takes a parameter of decimal named miles.
- Write on the board a class named Car that implements an interface named IDrivable.

Additional Resources

- Java OOP UDacity
  - https://www.udacity.com/course/object-oriented-programming-in-java--ud283
- Microsoft Docs
  - https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/object-oriented-programming
- Udemy
  - https://www.udemy.com/basics-of-object-oriented-programming-with-csharp/

Keep Practicing!

- Try creating new classes and instantiating them.
- Try to think of different parent child relationships and implement with inheritance.
- Try to think of classes that could implement an interface and create the interface and classes that implement it.