Logols Learning

WEEKEND WEB DEVELOPMENT BOOT CAMP

TRAINING: OBJECT ORIENTED PROGRAMMING

OOP – Object Oriented Programming

- Classes Defined and Separated Based Upon Properties and Methods
- Single Responsibility
- Code Reuse
- Inheritance
- Classes vs. Objects

Classes

- Statement block that contains:
  - Methods – Actions
  - Properties/Fields – Data
- Relates to an object in the Real World
- Has a Single Responsibility
- Becomes an Object when Instantiated

Class Syntax

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

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

Namespaces

- Statement block that contains classes
- Group Related Classes
- Similar to a Category
- Contain periods . to denote Sub Categories
- Often follow a naming convention like:

[Company].[Application].[Component].[Category]

Namespace Syntax

```
namespace [name]
{
   Statements…
}

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

Using Directive

- Allows use of Type in a Namespace
- Listed at the top of a code file above the namespace.
- Easier/Shorter than Listing a Type with the Namespace
- Example:

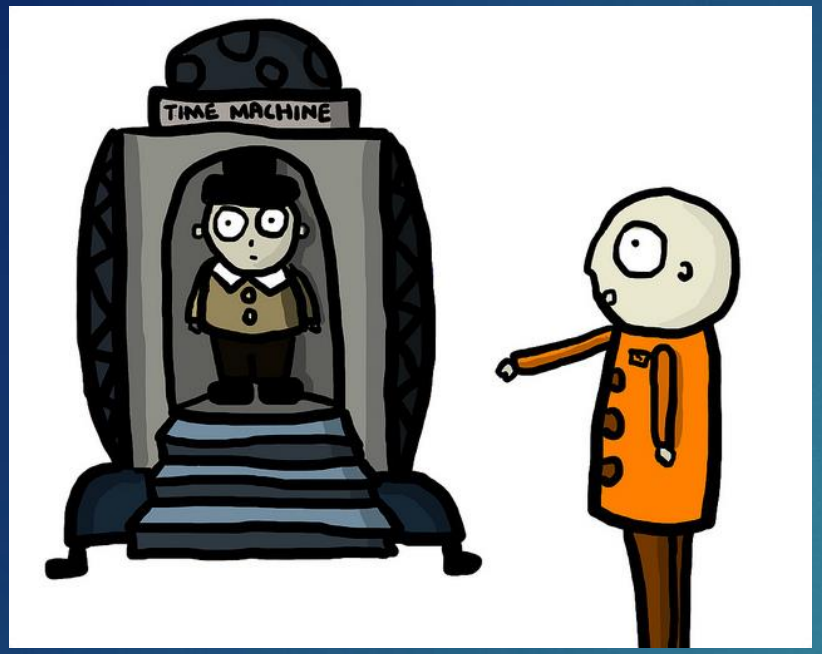
```
using LogoIs.Assessment.Entities.Subjects;
```

Constructor

- Method called when a class is instantiated
- Method Name = Class Name
- Return type or void is not used
- Can be overloaded

- Example:

```
public class Car
{
   public Car()
   {
      Statements…;
   }
}
```

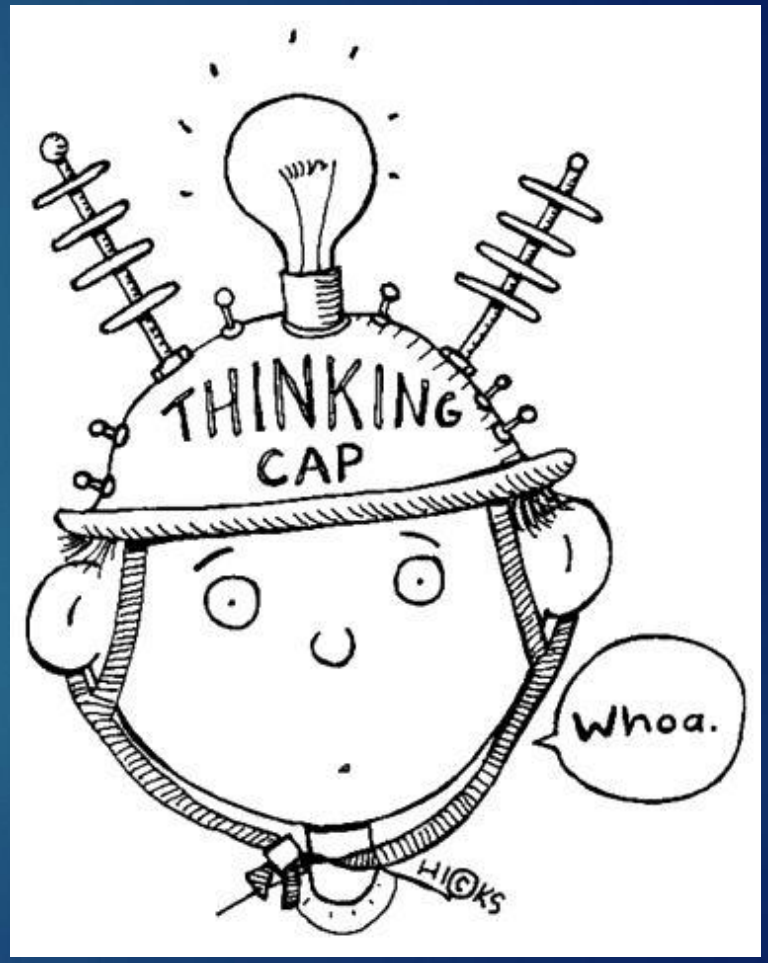


EXAMPLE

CLASS WITH CONSTRUCTOR

ASSESSMENT

CLASS WITH 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

- Data Associated with a Class
- Part of the Interface
  - Available to Other Classes
- Get – Allows retrieval of the data
- Set – Allows assignment of the data
- Get or Set may be left unimplemented

Property Syntax

General Syntax

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

Fully Implemented Example

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

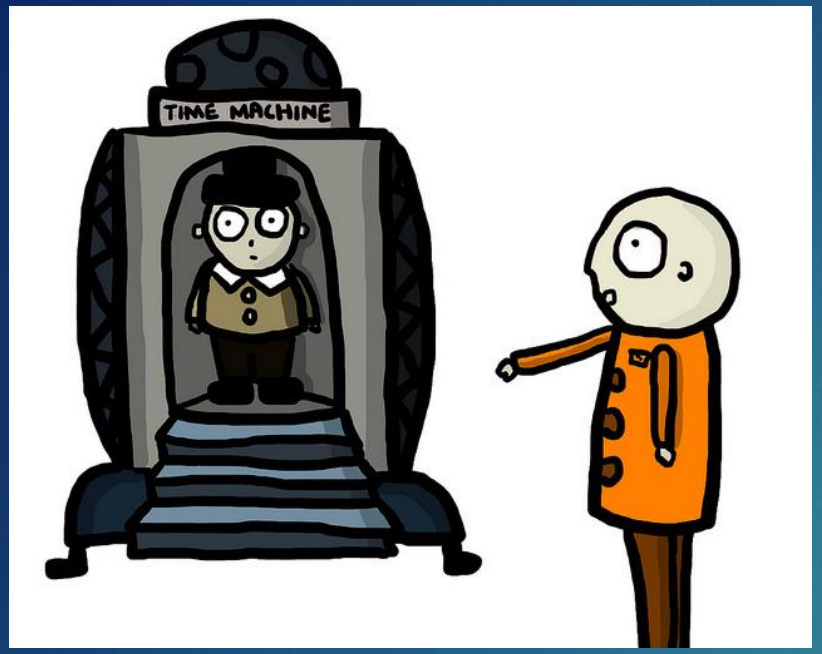
Auto Implemented Example

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

Instantiating and Using Objects

- A class needs to be instantiated to be used
- A class can be instantiated many times
- One instance of a class does not effect another
- Example:

```
Car car = new Car();
Console.WriteLine(car.DistanceTraveled);
car.Drive(15);
```

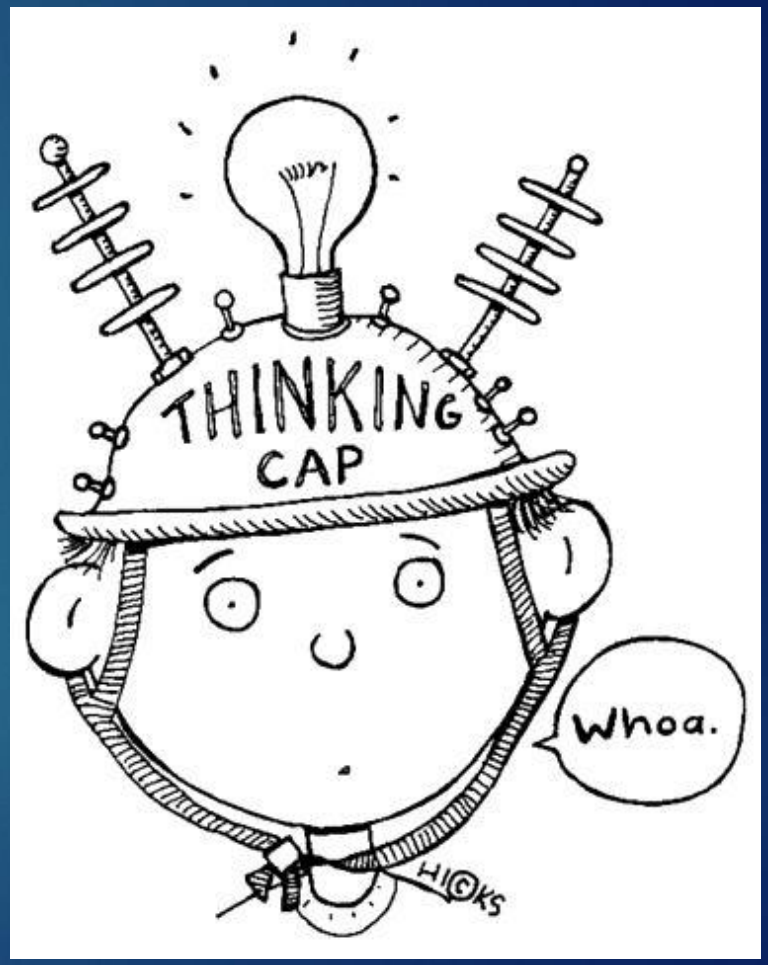


EXAMPLE

CLASS WITH PROPERTIES AND METHODS

ASSESSMENT

CLASS WITH PROPERTIES AND METHODS



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
- Usually used for one time use.
- var keyword
  - Declares a variable of unknown type.
- Example:

var person = new { FirstName = “Joe”, LastName = “Mackie” }

Encapsulation

- Only provide what's necessary
- Hide everything else
- Easier to use
- Less chance to incorrectly use class

Scope Access Modifiers

- public – accessible to everyone, not restricted
- Internal – access limited to current assembly
- private – access limited to defined class
- protected – access limited to derived classes

Variable Scope

- Public Variables
  - Available to other classes
  - Use properties instead
- Modular Variables
  - Available within the class
- Local Variables
  - Available within the method

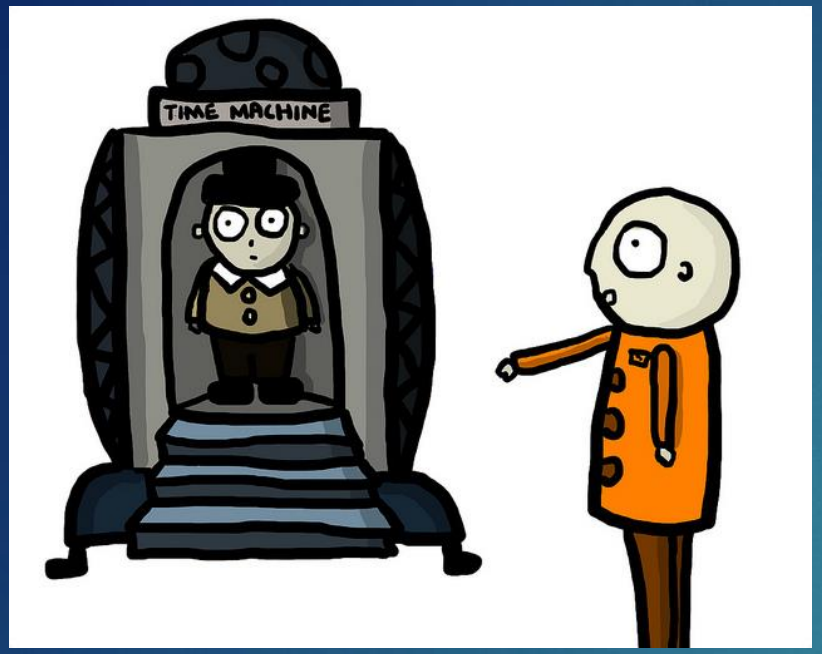
Inheritance

- Base Class
- Derived Class
- Example

Shape	Rectangle	Circle
• draw() • area()	• height • width • draw() • area()	• diameter • draw() • area()

Abstract, Sealed, Virtual, Override, Static

- abstract
  - forces the derived class to implement method
  - abstract class can only be derived cannot be instantiated
- sealed
  - prevent inheritance of class or method
- virtual
  - allows override of a method
- override
  - overrides an abstract, virtual, or override method

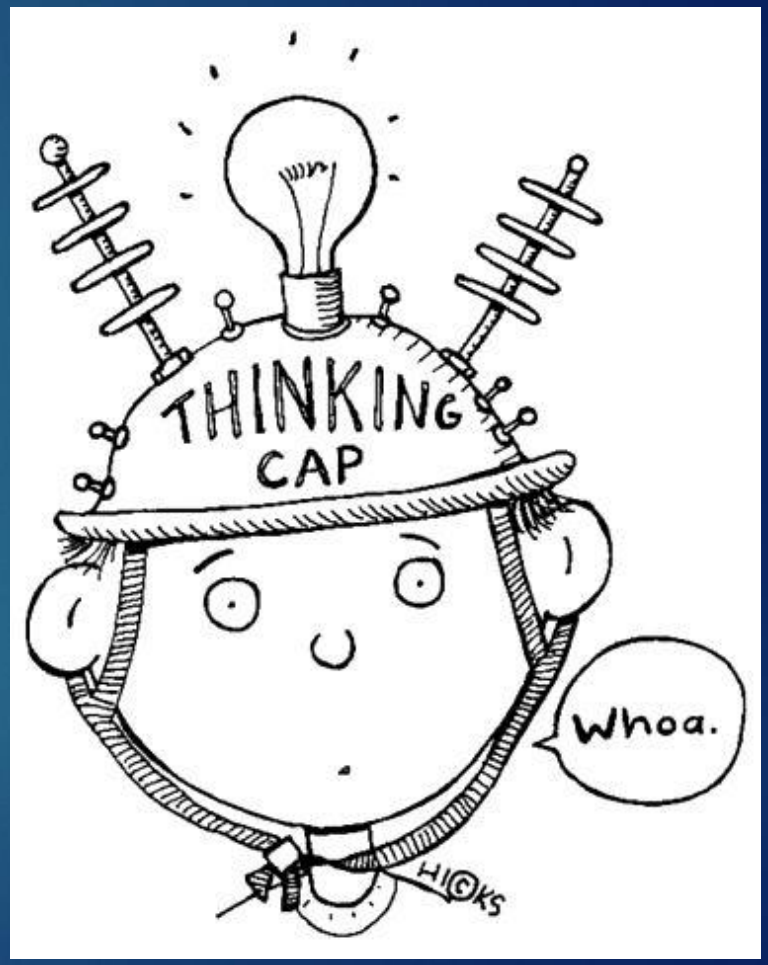


EXAMPLE

INHERITANCE

ASSESSMENT

INHERITANCE



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.

What is an Interface?

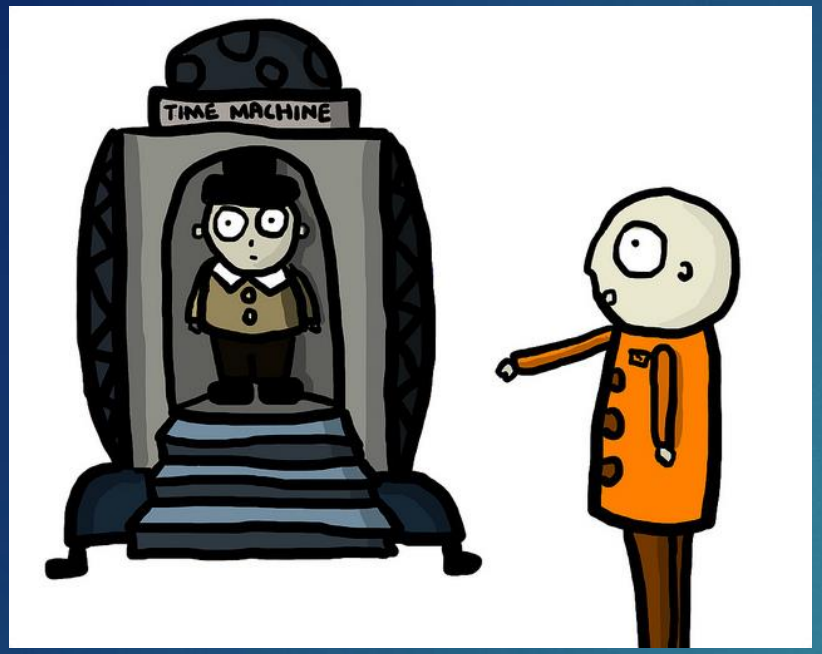
- Defines a set of properties and methods
- Contains no actual statements
- Classes and Structs can implement interfaces
- Classes can implement multiple interfaces
- interfaces vs. inheritance

Polymorphism

- Two different classes can be used in the same way.
- Implemented using interfaces or inheritance
- Variable declared as interface or base type

Example:

```
IDisposable stream = new Stream();
stream.Dispose();
```

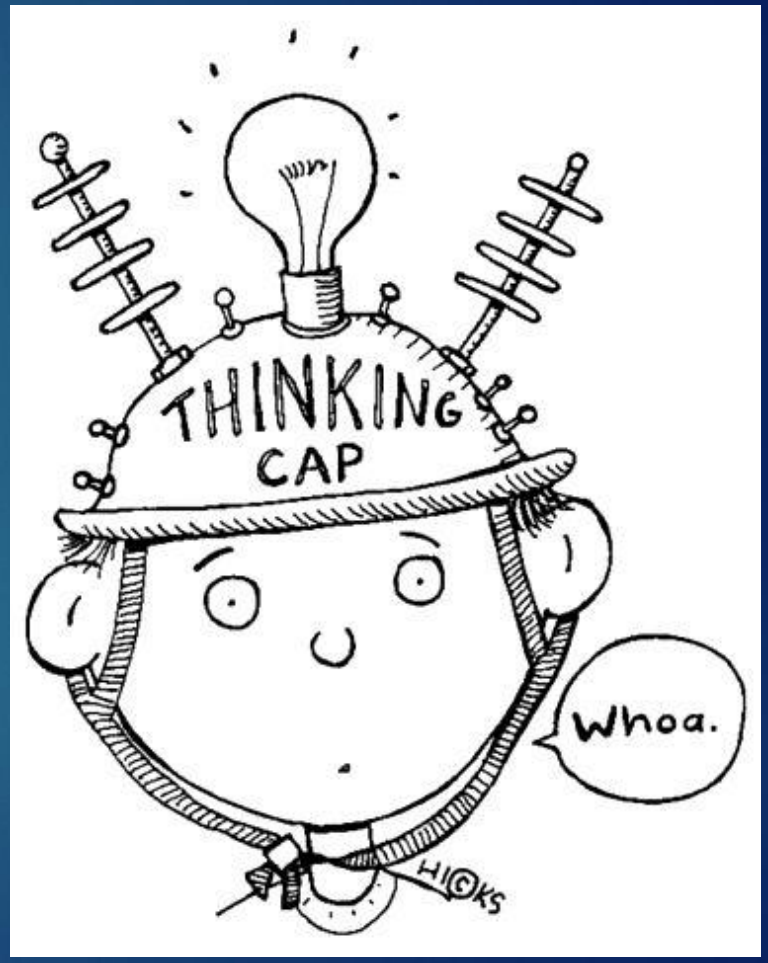


EXAMPLE

INTERFACES

ASSESSMENT

INTERFACES



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

Not really a sign you'd want to see whilst driving through an eerily quiet neighbourhood...

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.