Introduction to Classes

The Overview

Goals: Can you … ?

- ❏ ...explain the concept of classes and their use in Object Oriented Programming
- ❏ ...create a proper class definition
- ❏ ...effectively use class variables, methods and properties
- ❏ ...create and call Constructors
- ❏ ...describe the difference with `public` and `private` access modifiers
- ❏ ...create instances of a class
- ❏ ...explain the concept of properties and correctly define and use them in a class
- ❏ ...define member methods in a class and use them in an object of that class
- ❏ ...state the concept of overloading as it pertains to classes

Object Oriented Programming - OOP

Take a deep breath, the basics of OOP are as easy as PIE!

Three Fundamental Principles of OOP

Polymorphism - the ability for our code to take on different forms

Inheritance - the practice of creating a hierarchy for classes in which descendants obtain the attributes and behaviors from other classes classes.

Encapsulation - the concept of hiding values or state of data within a class, limiting the points of access

What is a Class ?

A class is a blueprint or model that defines ***state*** with **properties** (aka variables or fields) and ***behavior*** with **methods**. We create new instances of a class that follow the blueprint but may have different property values.

- Class Naming
  - Use *nouns or noun phrases*, not verbs
  - Try to use the *singular form* of a class name
  - Class name should match the file name
  - Follow Pascal Casing

Properties & Instance Variables

Properties define the state of an object:

A car can be any color such as blue or red and have an owner.

In that sense, an Object is said to have a property of *color* and *owner. They* are attributes of being a car.

Once a car leaves the factor, an owner cannot simply declare that their blue car is now red. The color is a private property of the car, it represents an internal or intrinsic value associated with that particular instance of a car. But the owner can sell it to someone else themselves.

Getters and Setters

Because the color of the car is an intrinsic or private property of the vehicle, it cannot be readily changed by the owner. Instead, they have to have it repainted using a special process by someone else.

In an Object, this is achieved using special methods called Getters and Setters.

- Getter: The customer does not care that their blue car is factory code B5, they know it as Ice Blue. A setter ensures that the private color is returned in an acceptable form using *getColor()*
- Setter: In order to change the color of the car, we can define a Setter Method *setColor()* that will perform the necessary work and change the internal (private) property color.

Constructor(s) … aka CTOR

Because a class is a plan or blueprint for an Object, we must have a way to build the actual object from that plan. This is the job of the **Constructor.**

For our car example, a class named Car must be built, the constructor is its factory.

Constructors take many forms and use parameters to determine what state the constructed (***Instantiated***) Object will have.

Consider: Car() myCar = new Car(“Blue”) -or- Car(“Blue”, “8cyl”) -or- Car(154003)

Methods: Actions & Behaviors

Methods control how an Object behaves and what actions it can take.

- **Private** Methods do work, perform actions, internally or within the Object and define an object's behavior
- **Public** Methods are actions that can be taken on an Object to change its state or trigger behaviors.

When you turn the ignition, you are performing an action on the Car, Car.start(). Internally, the car knows to close a circuit, spin a starter motor, and start the fuel pump and ignition .. .energizeCircuits(), .runStarter(), .pumpFuel()