Object Oriented Programming
(OOP)
**SCENARIO:** Tailspin Toys is developing a simple, robotic dog for small children. The dog will connect to a computer via a USB cable, and kids will be able to use a simple application to personalize the toy to their liking. Although the hardware is nearly complete, Tailspin has not yet created the software that owners will use to customize their toy dog. Victor has been hired to begin work on the software.

Initially, Victor will need to design a **class** to represent the dogs. Naturally, the class will be named Dog.

The toy—and therefore the **Dog class**—will need to keep track of the dog's name, age, and gender, and it will be able to bark, walk, wag its tail, and sit.
So far, he has written the following C# code

```csharp
'Creates the “Dog” Class / Object
public class Dog {
'Creates the Dog Method
public Dog() {
    name = "Hugo";
    age = 1;
    gender = 'M';
}
}
```
An object often represents something from the real world; in this case, the Dog class represents the robotic toy (or, it simply represents a dog).

In object-oriented design, verbs (such as barking, walking, tail-wagging, and sitting) are typically methods; attributes (such as name, age, and gender) are properties or fields.

A field is a variable declared within a class; a property provides a simple way to access the data stored in a field.
A class is like a blueprint. It defines the properties and methods that all objects of that class will have in the same way a blueprint defines the attributes of a house.

Multiple objects can be created from a class, or instantiated. Similarly, many houses can be built from one blueprint.

Objects are instantiated (created) with the new keyword.

We will now look at an example.