CGT 215 Lecture 3

Introduction to Classes and Objects

Car analogy

- Somebody tell me about the car analogy…

Car analogy

- Suppose you want to drive a car and make it go faster by pressing on the gas pedal.
- What must happen before you can do this?
- Before you can drive it, somebody had to design it.
- Designing a car typically begins with engineering drawings, or blueprints for ‘how’ to make a car.

Car analogy

- The blueprints include the design for the gas pedal, as well as the brake pedal.
- The gas pedal “hides” the complex mechanisms that actually make the car go faster.
- The brake pedal “hides” the mechanisms that make the car slow down.
- The steering wheel “hides” the mechanisms that make the car turn left and right.

Car analogy

- This enables people with little or no knowledge of how engines work to be able to drive a car.

- However, you can’t drive the blueprints – you have to build a car first.

Method

- Actions
- A method has () after the method name
- ***Performing a task*** in an application requires a method. The ***method*** describes the mechanisms that actually perform its tasks.
- The method hides from its user the complex tasks that it performs – just as a gas pedal would hide the tasks for making a car go faster.

Class

- In C#, we begin by creating an application unit called a ***class*** to house (among other things) a method – just as a car's blueprints house (among other things) the design of the gas pedal.
- In a class, you provide methods to perform the class's tasks.
- GoFaster() and SlowDown() might be methods of a car class.

Object

- Just as you cannot drive a blueprint of a car, you cannot “drive” a class.
- Just as you must build a car first, then drive it – you must build an ***object*** of the class before you can get an application to perform the tasks that the class describes.

Objects

- Briefly Defined
  - An object groups related methods, attributes, & properties
  - Reusable software components
  - A building block for you to use & reuse
  - Typically models something in the real world

Attributes

- A car has many ***attributes*** as well, such as its color, make, model, number of doors, amount of gas in the tank, current speed, and total miles driven (to name a few)
- These attributes are also a part of the car's design plans / blueprint – and always travel with the car as long as it exists.
- Every car maintains is own attributes

Attributes

- Every car maintains is own attributes
- For example, each car knows how much gas is in its own tank, but not how much is in the tanks of other cars.
- The same is true for each object you create from a class.
- These attributes are specified as part of the class.

Properties

- Notice that these attributes are not necessarily accessible directly.
- You don’t climb under a car, un-mount the gas tank, and look inside it to see if it’s full. You use the gas gauge on the dashboard.
- ***Properties*** are get and set accessors for reading and setting attributes.

Properties

- get accessors
  - For reading the values of variables (attributes)
- set accessors
  - For storing values into variables (attributes)

- GetCruiseSpeed();
- SetCruiseSpeed();

Instance Variables

- When you build a car, you are building an ***instance*** of that car. There may be 20,000 Ford Fusion cars on the road, but you only have an ***instance*** of that car.
- When you create an object from a class, it is called an ***instance variable***
- This process is called: ***instantiating an instance of an object***

Method Call

- When you drive a car, pressing the gas pedal sends a message to the car to perform a task – make the car go faster.
- Similarly, you send messages to an object – each message is known as a ***method call*** and tells a method of the object to perform a task.
- GoFaster(); //a method call
- SlowDown(); //a method call

Methods & Attributes

- Generally, () signifies a method
  - () distinguishes a method from an attribute
- Conversely:
  - An attribute is set with a value
    - x = 15;
    - isCorrect = true;
    - name = “Deitel”;
  - No code is executed other than the assignment of the value to the attribute

My Dog

Methods

Attributes

Properties

My Dog

- private Dog oMyDog = new Dog();

Methods

Run()

Eat()

Sleep()

Jump()

Bark()

RollOver()

Lick()

Age()

…

Attributes

name
gender
birthDate
breed
hairLength
hairColor
size
…

Why isn't 'Age' an attribute of oMyDog?

Properties

GetName()

SetName()

GetGender()

SetGender()

GetBirthDate()

SetBirthDate()

GetBreed()

SetBreed()

…

Declaration vs. Instantiation

- ***Declaration***
  - int x;
  - string name;
  - bool isCorrect;

- ***Instantiation***
  - `private Connection oConn = new Connection();`
  - `private Dog oMyDog = new Dog();`
  - `private Car oCar = new Car();`
  - `private Person myDad = new Person();`
  - `private Person myMom = new Person();`

Instantiation vs. Initialization

- ***Instantiation*** creates an instance of an object, as in the previous slide

- ***Initialization***
  - Assigning a value to a variable
    - name = “Deitel”;
    - x = 1;
    - isCorrect = false;

Access Modifiers

- private
- protected
- public

Access Modifiers: private

- A class's ***private*** variables and methods are not directly accessible to the class's clients. They are not accessible outside the class.

- Another way to say it: Variables, properties, and methods declared with access modifier ***private*** are accessible only to properties and methods of the class in which they are declared.

- private int x;

Access Modifiers: public

- The primary purpose of a ***public*** method is to present to the class's clients a view of the services the class provides (the class's public interface).
- Clients of the class need not be concerned with how the class accomplishes its tasks.
- ***public*** members are accessible wherever the application has a reference to an object of that class or one of its derived classes.
- public string name;

Access Modifiers (cont.)

- Note that members of a class – for instance, methods and instance variables – do not need to be explicitly declared ***private***.
- If a class member is not declared with and access modifier, it has ***private*** access by default.
- int y; //automatically declared private

Software Engineering Observation 4.2

- Precede every field and method declaration with an access modifier.
- Generally, instance variables should be declared private and methods and properties should be declared public.
- If the access modifier is omitted before a member of a class, the member is implicitly declared private.

Access Modifiers: protected

- Using ***protected*** access offers an intermediate level of access between *public* and *private*.
- A base class's ***protected*** members can be accessed by members of that base class ***and*** by members of it's derived classes.
- We'll discuss this more when we learn about inheritance.
- protected string gender;

Demo

- Create Dog class
  - Methods, attributes, property accessors
- Instantiate multiple dogs
- Make method calls
- Change attributes
- Change access modifiers