JS 14 Classes & Objects

Projects 1 - 13 was about creating variable and then writing functions as actions. **Is there a way for us to combine variables and functions together**? Yes we can and combining them together creates a class.

In this session, we focus on how to combine variables and functions together to **create a class** and then how to access them using the **member access operator**.

1. public vs private 2. member access operator 3. classes vs objects

==========

1. Public Versus Private

The difference between **public** and **private** is the number of owners. A public entity has many owners while a private entity has a single or low number of owners.

In programming, we also use the idea of public and private to customize our programming solution so that it can satisfy the problem statement.

Public

```
var fName = "Vuong";
```

The code above declares a variable called **fName** and loads the data "**Vuong**" into it. Also, note the assignment operator (=) and delimiter (;)

Private

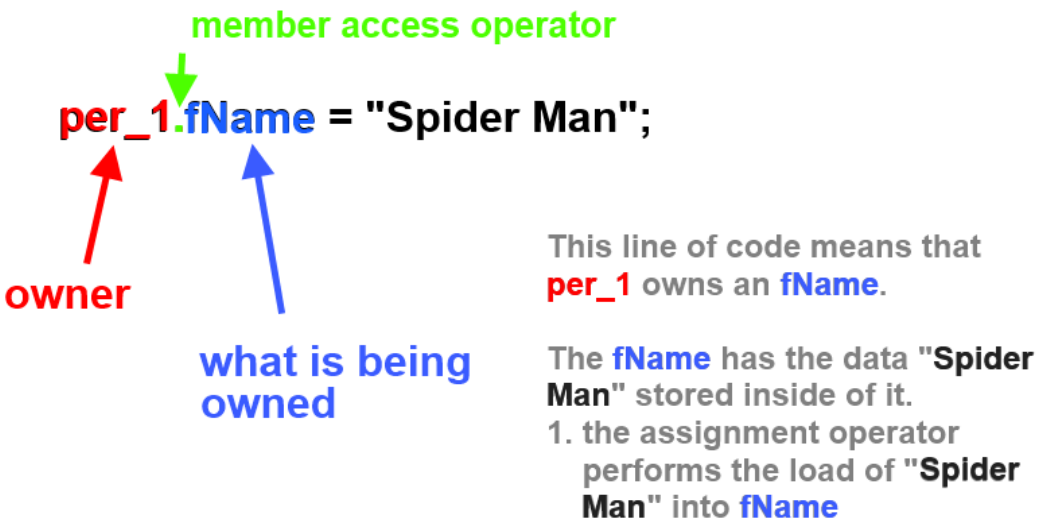
```
// -- Class definition
class Person
{
	constructor()
	{
		this.fName = "Vuong";
		this.lName = "Nguyen";
	}

	getFName()
	{
		return this.fName;
	}

	getLName()
	{
		return this.lName;

	}

}

// -- using "new" to create an object named "per_1"
var per_1 = new Person();


per_1.fName = "Spider Man";
```

Explanation

Look at the period for the line of code `per_1.fName = "Spider Man";` This is important because the period is **called the member access operator and it represents ownership**. When you own something, it belongs to you and only you have access to it. This is called private.

To the **left side** of the member access operator is the **owner** and to the **right side** of the member access operator is **what is being owned**.

Continue to the next page



Compare Public Vs. Private

```
var fName = "Vuong";                    // -- public
```

The line of code above **DOES NOT** have the member access operator and this means lack of ownership. A lack of ownership means it is a **public (or global variable)** and everyone is the owner of that variable. If everyone is the owner, then everyone can use it.

```
per_1.fName = "Spider Man";          // -- private
```

The line of code above **DOES** have the member access operator. This means that `per_1` owns an `fName`; This means that ONLY `per_1` has access to `fName`.

2. **Review Challenge**

```
fName = "Code E";

per_1.fName = "STEMA";
```

1. What is the data that is stored inside the public variable `fName`?
2. What is the data that is stored inside the private variable `per_1.fName`?

3. Classes Vs Objects

In a real world situation, there can be multiple objects of the same type. For example, we can have a single Honda Accord **model** but each Honda Accord **copy that is produced** can be customized by changing its color.

So, there is a single Honda Accord model (**a class**) but we can produce many Honda Accords (**objects**) with different colors, like silver, red, green, blue, white, black, and etc.

The single Honda Accord model is a **generic template** and this generic template is called a **class**. What makes a model generic? There are **default values** that serve as a reference point.

```
class Person
{
	constructor()
	{
		this.fName = "Vuong";
		this.lName = "Nguyen";
	}

	getFName()
	{
		return this.fName;

	}

	getLName()
	{
		return this.lName;

	}

}
```

The generic template is named Person.

1. **This generic template has a default value of "Vuong" as the fName.**
2. **This generic template has a default value of "Nguyen" as the lName**

We can **clone the generic template** to make multiple, **specific copies** and these copies are called **objects**. What makes them specific? Once we make a copy (or an object), we can then customize its color to be anything we like.

```
var per_1 = new Person();
var per_2 = new Person();
var per_3 = new Person();
```

We create multiple copies of the generic template by using the word "new".

Here we are creating 3 copies of the generic template ➔ we are making 3 objects

1. **The specific copies are called "objects"**
2. **Once we make 3 copies, we can customize their property values by using the assignment operator**

```
per_1.fName = "Spider Man";

per_2.fName = "Bat Man";

per_3.fName = "Cat Woman";
```

Format: nameOfObject.nameOfWhatIsOwned

This code above uses the format and translates into

"per_1 owns fName"

If we are on the **outside of a class definition**, then we use the **format above**.

What if we are on the **inside of a class definition**? The next chapter will talk about the use of **this.** as a shortcut.

Continue to the next page

4. Write The JS Code In Between <script> </script>. Write ALL OF IT!!! Color code is for explanation.

```
class SongProfile
{
      constructor( bName, sGenre )
      {
            this.bandName   = bName;
            this.bandId     = "bandName";

            this.genre      = sGenre;
            this.genreId    = "genre";
      }

      show_profile()
      {
            document.getElementById( "bandName" ).innerHTML = this.bandName;

            document.getElementById( "genre" ).innerHTML    = this.genre;

      }

}
```

Explanation

Creating the class starts with the reserve word "**class**". Reserve means that it has a special purpose and is only reserved for that special purpose. After that, we give the class a name. In the code above, the class is named "**SongProfile**".

How do we create private variables? We use a **constructor,** whose purpose is to create private variables and give the newly created private variables default values. Inside our constructor, we have 4 private variables and they all begin with **this.**

1. If we are on the **inside of the class definition**, **this.** is used as a **shortcut** to represent the owner
2. We could write SongProfile.bandName **OR** We could use **this.** as a shortcut ➔ **this.**bandName.

this. represents the **first person view**. If I own a cell phone, I don't say "Vuong owns a Samsung Phone". Instead, the sentence would be "I own a Samsung Phone".

We can also have **private function definition**. Inside the class definition, we have one private function definition and it is called **show_profile().**

Notice that **public** function definitions start with "**function**" while **private** function definitions **do not** have the word "**function**".

Public Function Definition

```
function get_sum( num1, num2 )        // -- has "function"
{
        return num1 + num2;

}
```

Private Function Definition

```
show_profile()                        // -- does NOT have "function"
{
      document.getElementById( this.titleId ).innerHTML = this.title;

      document.getElementById( this.genreId ).innerHTML = this.genre;

}
```

Continue To Next Page

5. Write The JS Code In Between <script> </script>. Write ALL OF IT!!! Color code is for explanation.

```
function init_system()
{
	alert( "inside init_system()" );

	/* ==================
	bName, sTitle, sCaption, sGenre, sFN
	*/
	 var song_0 = new SongProfile( "Magic!", "Pop & Stuff" );


	song_0.show_profile();

	playlist.push( song_0 );


	trackNum = 0;

}
```

Explanation

The important part is the line of code that is highlighted in **purple** and **gold**. The word "**new**" means that we are making a copy of a class. Everytime we make a copy of a class, we get an object.

In the above, the object is called `song_0`. What makes a class different from an object? Objects are specific and can be customized. The code in **green** customizes `song_0` to have "**Magic!**" as the **band name** and "**Pop**" as the **genre**.

Look back up to chapter 1 and you will see that the green highlighted colors match by position. So, "`Magic`" is stored into `bName` and "`Pop`" is stored into `sGenre`.

We have a playlist that is an array and we put the object named `song_0` at the end using `.push().`

Third Person View

Go back to chapter 4's explanation and briefly read about the **first person view**. Is there a 3rd person view? Yes, and the object is our 3rd person view.

The format for 3d person (or object view) is: **objectName.whatObjectOwns**

To change the bandName to “The Jackon 5”, we write

```
song_0.bandName = “Jackson 5”;
song_0.genre    = “Mo Town”;
```

1st Persion View – Inside Class definition ➔ use this. as a shortcut to represent the owner

```
class SongProfile
{
      constructor( bName, sGenre )
      {
          this.bandName = bName;
          this.genre    = sGenre;

      }
}
```

3d Person View – Objects ➔ outside of class definition. Use name of object

```
song_0.bandName = “Jackson 5”;
song_0.genre    = “Mo Town”;
```

Continue to the next page

6. Write The HTML Code In Between <body> </body>. Put the code below the <body> tag. Write ALL OF IT!!! Color code is for explanation. Position the code under the code in Purple

```
<body
id      = "bTag"
onload = "init_system()"
>

<audio
controls
id = "mp3Tag" src="blah" type="audio/mpeg">

      Your browser does not support the audio element.

</audio>
```

Explanation

This is our first time using the <audio> </audio> audio tag. Remember the image tag <img/>? The <audio> </audio> tag and <img/> tag both have the "src" attribute. The "src" is the location of the image or audio file on the internet.

Next, give the audio tag an id of "mp3Tag". This is important because giving an id to an HTML element allows JS code to link up with it.

Finally, we have the "controls". This means that the play, pause, and volume controls are displayed.

7. Write The CSS Code In Between <style> </style>

```
#songInfo table#playlistTb tr td
{
      padding    : 0px 0px 0px 0px;
      margin     : 0px 0px 0px 0px;


      width      : 50px;
      height     : 50px;

      /*background-color: #909090;*/


}
```

```
body
{
	background-color: #a7a7fa;


}
```

Explanation

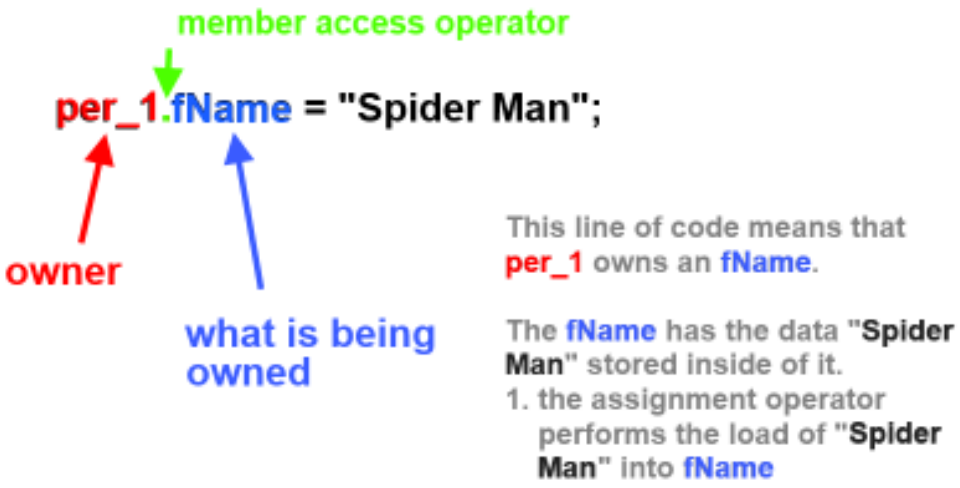
We are styling the table's data cell. Remember that **tr** stands for **table row** and **td** stands for **table data**.

HTML Challenge 1

1. make a paragraph tag and give it an id of "**genreColor**". Next, attach a click event of `onclick`. When the user clicks on this paragraph tag, the JS function "**change_bg()**" is called

JS Challenge 1a

1. Inside the function definition "**change_bg()**", check if the genre of song_0 is "**pop**".
    a. **HINT: the genre is a private attribute and song_0 is the owner**



2. If the genre of song_0 is "**pop**", then do the following
    a. Link JS code with the HTML element whose id is "**bTag**"
    b. Change the **backgroundColor** to "#006500"

    HINT: DO NOT user `.innerHTML` because it is used to change the data of the website. Instead, we are changing the style so we use `.style`.

JS Challenge 1b

1. Inside the function definition "**change_bg()**", check if the genre of song_0 is "**motown**".

HINT: the genre is a private attribute and song_0 is the owner

a. If the genre of song_0 is "**motown**", then do the following
b. Link JS code with the HTML element whose id is "**bTag**"
c. Change the **backgroundColor** to "#f7a8fa"

HINT: DO NOT user `.innerHTML` because it is used to change the data of the website. Instead, we are changing the style so we use `.style`.

JS Challenge 2

Go back **INSIDE** the class definition of **SongProfile**, Go back to Chapter 4 for reference

1. Add another private attribute and name it "**caption**"
2. Add another private function and name it "**get_caption()**"

JS Challenge 3

Go back inside to the function definition of `init_system()` (go back to chapter 5). Next, use the 3rd person view and do the following

1. load the data "**good music**" into the private attribute **caption.** Remember, **caption** belongs to **song_0**
2. call the private function "**get_caption()**". Remember, "**get_caption()**" belongs to **song_0**