GD 6
Words In Space

So far, we have learned about **HTML 5 Canvas** and the two different levels that make up modern video game development.

What is a game object? **A game object is a clone of a class**. A class is a generic model that can be cloned to make duplicates. We are going to use the **new** operator and the **constructor()** to create multiple clones of the QA class. We then customize the clones with different question and answers. In this lesson, we combine everything together to create a **basic educational, side scrolling game**.

Words In Space – match the answer with the questions. Control your shape ship using the computer mouse.

1. Control Space ship using mouse
2. Side scrolling game
3. OOP

---

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 (;)

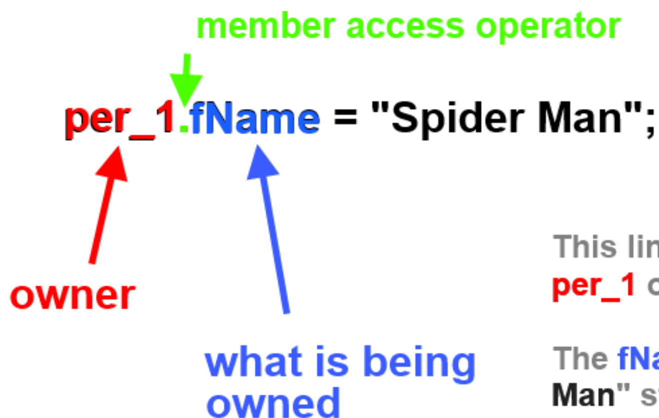
Private

```
// -- 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 **green** 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**.



This line of code means that **per_1** owns an **fName**.

The **fName** has the data "**Spider Man**" stored inside of it.

1. the assignment operator performs the load of "**Spider Man**" into **fName**

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();



per_1.fName = "Spider Man";

per_2.fName = "Bat Man";

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

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**

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.

4. SAMPLE ONLY!!!

```
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;

}
```

1. Write the code below in between <script> </script>. The **large, green banner** is your landmark. **Go to the coding website and look for it.** Next, write the code below underneath the **large, green banner**. **Write all of it, color code is for explanation.**

```
/* ================================
* QA Class Definition
*
* ================================ */
class QA
{
        constructor( question, answer)
        {
            this.quest = question;
            this.an    = answer;
        }

        check_if_matching( uAnswer )
        {
            return (this.an == uAnswer) ? 1 : 0;

        }

        get_qa_pair()
        {
          return { q: this.quest, a: this.an };

        }

}
```

Explanation

The code above is the class definition of QA, which holds our question and answer. Notice that the reserve word "`class`" is used to show that it is a **special way to combine attributes and functions together**.

Remember that a **class is a generic model** while an **object is a clone of a class**. Once we make a clone using a constructor, we can then customize the clone by giving the constructor parameters.

The class definition has two private attributes and they are called `this.quest` and `this.an`. Notice the reserve word "`this`" means that it is private property and this means that only the class has it.

The answers that are displayed onto the screen while falling with the asteroid come from the private property `this.an`.

2. Write the code below in between <script> </script>. The **large, green banner** is your landmark. **Go to the coding website and look for it.** Next, write the code below underneath the **large, green banner**. **Write all of it, color code is for explanation.**

```
/* ========================================
* Create QA object
*
* ======================================== */
var qa = new QA( "This stores data", "variable" );
allQAArr.push( qa );

qa = new QA( "Sending a message", "broadcast" );
allQAArr.push ( qa );

qa = new QA ( "Side to side movement", "x position" );
allQAArr.push ( qa );

qa = new QA ( "Up and down movement", "y position" );
allQAArr.push ( qa );

qa = new QA ( "Ends a line of code", "semi-colon" );
allQAArr.push ( qa );
```

Explanation

The code above uses the reserve word "new" to create multiple clones of the QA class. Once a clone is created by the word "new", we then load the clone into a variable named "qa".

Notice that the constructor takes in two arguments to match our constructor definition from above. Since objects can be customized, we can add different data to each clone.

Finally, we put the clone into an array using the `.push()` method. The items in the array will be randomly picked and then displayed onto the screen.

CONTINUE TO THE NEXT PAGE

3. Write the code below in between <script> </script>. The **large, green banner** is your landmark. **Go to the coding website and look for it.** Next, write the code below underneath the **large, green banner**. **Write all of it, color code is for explanation.**

```
/* ============================================
* Check if laser touched hit box of asteroid.
* Check if answer matches question
* ============================================ */
var tail = onscreenList.length-1;
for( var c = 0; c < cQALim; c++ )
{
if( p1LArr[laz].check_hb_of_circle( cArr[c].cXPos, cArr[c].cYPos, cArr[c].cRad )  )
	{

	var testObj = allQAArr[cArr[c].qaID].get_qa_pair();
	if( chosenQAObj.a == testObj.a )
	{
		console.log( " ....... correct answer to question ........" );
		// -- hit laser on ball, more laser
		laserLim++;

		// -- random asteroid shown if user is correct
		cQALim += Math.floor( Math.random() * 10 ) + 4;
		if( cQALim >= 9 )
		{
			cQALim = 6;
		}


		p1Score += 1;
		//onscreenList.pop();

		// -- reset circle to top
		cArr[c].reset_to_top();
		cArr[c].update_qa_id( Math.floor( Math.random() * allQAArr.length )
);

	}
	else // -- wrong answer
	{

	}


}

// -- move to next item on the left
tail--;

}
```

Explanation

The code in red uses the private function `.check_hb_of_circle()` to check if the laser has touched an asteroid. If true, then the code in **orange** will check if the answer from the asteroid

matches the answer of the question that is displayed on the top left. This is how we check if the player correctly matched the answer with the questions.

In reality, the question being displayed is one object, called `chosenQAObj`, and the asteroid the laser touched, called `testObj`, is a different object. We compare the private attributes of `chosenQAObj` with `testObj` and if their answers match, then the player chose the correct answer.

If the player chose the correct answer, then we execute the code in **gold**. It will give the player back the laser that was spent by incrementing the laserLim by 1, `laserLim++.` Next, we randomly pick a number and this number controls the amount of asteroids that are displayed onto the screen.

Finally, the code in **green**. Since the player got the correct answer, we reward the player with a single point. Next, we reset the asteroid to the top of the screen so that the object can begin a new life. We complete the process by assigning the asteroid a new **qa object**. Since we don't want the player to be able to predict what is coming next, we will randomly choose a **qa object** and store this number into a private variable called **qaId**. This links up the circle with a qa object.

CONTINUE TO THE NEXT PAGE

4. Write the code below in between <script> </script>. The **large, green banner** is your landmark. **Go to the coding website and look for it.** Next, write the code below underneath the **large, green banner**. **Write all of it, color code is for explanation.**

```
/* ===============================================
* Asteroid with answers falling down
*
* =============================================== */
var tail = onscreenList.length-1;
for( var c = 0; c < cQALim; c++ )
{
        ctx.fillStyle = "#" + cArr[c].cColor;
        cArr[c].move_one_step();
        if( cArr[c].check_vb() )
        {
             cQALim += Math.floor( Math.random() * 10 ) + 4;
             if( cQALim >= 9 )
             {
                    cQALim = 6;
             }

}

ctx.beginPath();
ctx.drawImage( asteroidImgSrcArr[cArr[c].imgIndex], cArr[c].cXPos, cArr[c].cYPos, cArr[c].cRad, cArr[c].cRad );

ctx.fill();

        // -- draw text
        ctx.beginPath();
        ctx.fillStyle = "#" + cArr[c].cColor;
        ctx.font  = "Bold 20px Arial";
        var qaObj = allQAArr[cArr[c].qaID].get_qa_pair();
        ctx.fillText( qaObj.a, cArr[c].cXPos, cArr[c].cYPos+20, 70 );
        ctx.fill();

        // --
        tail--;
}
```

Explanation

The code in **red** is important because it loops through and redraws every asteroid object in their updated (x,y) position. Remember that the screen is wiped and we must redraw every game object in their update (x,y) position every 16 milliseconds. We stored our asteroid object into an array called cArr and we begin at the first position of 0.

We use a digital key of **c** so that we can jump from position 0 to position 1 to position 2 and etc. The code **c++** allows us to jump by 1 position.

The code in **blue** changes the position of the game object by moving it down 1 step on the y axis. The code in **green** then checks if the asteroid has touched the bottom boundary. If so, then we will reset the asteroid to the top and then link it up with a new qa object.

The code in **red**, **blue**, and **green** updated the (x,y) position of the asteroid and qa objects. This is level 1. Now, we must redraw again and the code in orange and gold does that.

The code in **orange** redraws the asteroid with a new costume. The code in **gold** redraws the answer as text and this is why `ctx.fillText()` is used.

JS Challenge 1

1. Create a digital key and name it "**p**"
2. Use a "**for**" loop and loop through until the end of the array **playerArr**
3. Jump by **1 position**
4. Select a single position from the array and load the data into a variable called pNum
5. Check if the pNum is equal to 4 and send an alert if it is true. The message of the alert should be "you got it!"

JS Challenge 2

1. What is a class?
2. What is an object?
3. What operator is used to create an object?
4. What operator is used to show ownership?

JS Challenge 3

Assume that you have a class definition **Person**.

1. Create a variable named **Person1**.
2. Next, use the "**new**" operator to create an object of type **Person**.
3. Next, use the **assignment operator** to load the newly created object into the variable **Person1**.
4. Translate this code into plain English: Person1.basketball
5. Load the data "**Spalding**" into Person1.basketball