GD 17
Penalty Kick

We have seen that making a copy of a class definition creates an object. We use the class definition as the original and then use the "new" operator to create clones of the original (which becomes objects).

In this lesson, we explore the different ways of writing code for a class definition versus object.

Penalty Kick is a guess game where we try to kick the ball to a spot to score a goal. The challenge is that there are 4 spots and the goalie could also pick a spot that we choose. Due to the 4 spots, there is a 25% chance the goalie blocks our shot.

1. class definition 2. Constructor 3. "this." versus object name

---

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.**

```
/* ======================================
* Zipline Constructor
* ====================================== */
constructor( xPos1, yPos1, xPos2, yPos2, stepSize )
{
      this.xStPos   = xPos1;
      this.yStPos   = yPos1;
```

Explanation

The code above uses a **constructor** to initialize the Zipline class. When it comes time to create clones (ie. objects) and we wanted to customize the object, the **inputs to the open and closing parenthesis** are values that are used to customize the object.

Inside the body of constructor, we created two private variables and they are named this.xStPos and this.yStPos. Next, we loaded the input xPos1 into this.xStPos and input yPos1 into this.yStPos.

JS Challenge 1 - Add onto the class definition and add the following private variables. Remember that private variable use "`this.`"

Look for the green banner below and put the code under the banner

```
/* ===================================
* JS Challenge 1
* =================================== */
```

1. private variable named **xEnPos** and use the assignment operator to load the input xPos2 into the private variable

2. private variable named **yEnPos** and use the assignment operator to load the input yPos2 into the private variable

3. private variable named **stepSize** and use the assignment operator to load the input stepSize into the private variable

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.**

```
/* ========================================
* Move One Step
* ======================================== */
move_one_step()
{
    this.opcode = ( this.xStPos < this.xEnPos ) ? 1 : 0;

    this.mouseYR = this.yStPos - this.yEnPos;
    this.mouseXR = this.xStPos - this.xEnPos;

    this.mouseTR = this.mouseYR / this.mouseXR;

    // -- fly to top right
    if( this.opcode )
    {
        // -- move line, start point
        this.xStPos   += this.stepSize;
        this.yStPos   += this.stepSize * this.mouseTR;

    }
    else // fly to top left
    {
        // -- move line, start point
        this.xStPos   -= this.stepSize;
        this.yStPos   -= this.stepSize * this.mouseTR;

    }

}
```

Explanation

The function above is a private function because it does not have the word "**function**" in front of the function name.

```
// -- public                          // -- private
function move_one_step()              move_one_step()
```

Another **important difference** is that the private function is **inside the curly braces** of the class definition of `class Zipline`.

The private function checks to see the direction of the line and will then command the ball to fly in that line. The ball can fly in the top right direction or top left direction.

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.**

```
/* ======================================
* Using Private Function .move_one_step()
* ====================================== */
if( zipLineArr[z].check_end_hb( zipLineArr[z].xEnPos, zipLineArr[z].yEnPos ) )
{
      zipLineArr[z].load_new_step_size( 0 );
}
else
{
      zipLineArr[z].move_one_step();
}
```

Explanation

In the previous step, we defined the private action `move_one_step()`. In this step, we will use it by calling the name of the private action.

The important part of the code is the **orange code**. Remember that move_one_step() is a private function because it is **INSIDE the curly braces of class Zipline**.

When we are inside the class definition (ie, inside the curly braces), using a private function only requires using **this.**move_one_step().

1. Remember, the period "." shows ownership and it is called **the member access operator**
2. "**this.**" would mean that Zipline is the owner of move_one_step()

What if we wanted to use the private function outside of the curly braces? For example, we create an object outside of the curly braces of class Zipline and the object wants to use the private function move_one_step().

Since we are outside of the curly braces of class Zipline, we can NO LONGER use "this."

but now must use the nameOfObject.move_one_step(). The code in orange above is the example. The name of the object is `zipLineArr[z]` and it uses the member access operator to show ownership of `move_one_step()`.

Another example is

```
var zLObj = new ZipLine( 500, 300, spotMarkerArr[userSMIndex]-20, 120, 3 );

zLObj.move_one_step();
```

We used the **new operator** to create an object and the object is named **zLObj**. Since we are outside of the curly braces of class Zipline, we just the name of the object instead of "**this.**"

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.**

```
/* ========================================
* Penalty Spot
* ======================================== */
// -- larger, outer circle
cContext.beginPath();
cContext.fillStyle   = "#D8F5FF";
cContext.arc( 500, 300, 40, 20, 45, false );
cContext.fill();

// -- smaller, inner circle
cContext.beginPath();
cContext.fillStyle   = "#51D3FF";
cContext.arc( 500, 300, 25, 20, 45, false );
cContext.fill();
```

Explanation

The penalty kick game has a designated spot to kick the ball. The spot has an outer circle and an inner circle. The code above draws the larger outer circle and then the smaller inner circle. We are revisiting the cContext.arc() function to draw a circle.

You are free to change the colors of both circles.

JS Challenge 2 – This JS Challenge 2 will review class definition and accessing private variables and action.

Go to the website https://www.w3schools.com/code/tryit.asp?filename=GD76D4FQPT8Q to type the code.

Above function main()

1. write a class definition for **FoodItem**
2. the **constructor** will have 2 inputs
    a. **name**
    b. **color**
3. inside the **body of the constructor**, do the following
    a. create a private variable named **foodName** and load the input "**name**" into it
    b. create a private variable named **foodColor** and load the input "**color**" into it
4. inside the class definition, create a private function named **alert_color()**
    a. This private function will send alert of the color

Inside function main()

1. Create a variable named **foodObj**
2. Use the "**new**" operator to create an object of class **FoodItem**.
    a. **The first input is "Guava"**
    b. **The second input is "green"**
3. We are outside of the curly braces of class Food Item.
    a. Call the private function **alert_color();**