JS 21 Calendar OOP

Your friend "valid input check" is back again. In JS 20, we were able to modify the content of an event. We changed the event title, message, and priority level.

What about the situation where the event day has to be moved to a different day? For example, what if we have to move an event from Jan 5th → Jan 10th? This is the more difficult case because we have to delete the Jan 5th event from our website and then update our website by adding the moved event to Jan 10th.

In this session, we continue to focus on admin control by implementing "valid input check" and the movement of an event to a different day.

1. Add and remove 2. Valid input check

==========

1. **Write The JS Code In Between** `<script> </script>`. **ONLY** write the code in **PURPLE** and position it under the code in **BLACK.**

```
class EventProf
{
    constructor( mId, eDay, eMsg, uPri, hPri )
    {
        // -- eColor ==> event color
        this.eColor = this.get_cc( uPri, hPri );
        this.msg    = eMsg;
        this.day    = eDay;
        this.tabId  = mId;
        this.prevDay= 0;
```

Explanation

The code in **PURPLE** is a **private variable**. We know that it is private because it has **this.** in front of the name **prevDay**. If it is private, then who owns it? The class `EventProf` owns the private variable **prevDa**.

Remember that private ownership is represented by the member access operator (.). If we are inside the class definition, the word **this.** is all that is needed. Why? If I own something, I don't need to say "Vuong owns prevDay". Instead, the phrase would be "I own prevDay".

However, **when we are making an object**, then “Vuong owns prevDay” is correct.

```
var vuong = new EventProf();  // -- vuong is an object

vuong.prevDay = “5”;      // -- vuong owns prevDay
```

2. **Write The JS Code In Between <script> </script. ONLY** write the code in **SKY BLUE** and position it under the code in **BLACK**.

```
pop_form()
{
    document.getElementById("nEDay").value   = this.day;
    document.getElementById("nEInput").value = this.msg;
    document.getElementById("nETitle").value = this.eTitle;

    // -- set previous day to delete old occurance
    this.prevDay = this.day;
```

Explanation

Remember that when we click on an event block on our website, a pop up banner appears to show more information. We can double down on this click event. When we click on an event block on our website,

1. We show the pop up banner
2. We activate our admin control

We have to prepare for the case that the user wants to move an event from a day to a different day. Preparing to move an event block to a different day means that we must save the old day into **this.prevDay**, which was created in chapter 1.

We have to save the old day because we are going to use it as an id so that JS can find it. Once found, JS will remove it from our website.

3. Write The JS Code In Between <script> </script>. WRITE ALL OF IT!!!
Color code is for explanation. *The code below is a private function of EventProfile …*

```
update()
{
	//alert( "inside private fx update()" );

	var msg	= document.getElementById("nEInput").value;
	var uPri	= document.getElementById("uPri");
	var hPri	= document.getElementById("hPri");
	var title	= document.getElementById("nETitle").value;

	// --
	var day	= parseInt( document.getElementById("nEDay").value );
	//alert( "day: " + day );

	// -- same day, just update title and message
	if( day == this.prevDay )
	{
		//alert( "same day" );

		this.eColor = this.get_cc( uPri, hPri );
		this.msg	= msg;
		this.day	= day;
		this.eTitle = title;

   var eObj = document.getElementById( this.tabId + "_" + this.day + "_" + this.eTicket );

		//alert( "this.eColor: " + this.eColor );
		console.log( "this.eColor: " + this.eColor );



		eObj.innerHTML = this.eTitle;
	}
	// -- different day, remove old row and put data in new row
	else
	{
		//alert( "different day" );

		// -- remove old row
    var oldObj = document.getElementById( this.tabId + "_" + this.prevDay + "_" + this.eTicket );

		oldObj.remove();

		this.eColor = this.get_cc( uPri, hPri );
		this.msg	= msg;
		this.day	= day;
		this.eTitle = title;

		// -- make a new row
		this.make_clone( this.eTicket );

		// --
		this.hide_pop_up();

		/* -- new location means new HTML element was added on the fly
```

```
            a. add event listener on this new location
             */
             this.set_event_listener( this.eTitle );


        }
}
```

Explanation

There are two situations that we have to account for. The **first situation** is when the user only changes the title **OR** message of the event. The **second situation** is when the user changes the day **OR** title **OR** message of the event.

In the **first situation**, the day of the event is left unchanged and we only change the title or message of the event. If this is the case, we simply update the website by using `document.getElementById()` and the id of the event to allow JS code to link up with HTML. Next, we use `.innerHTML` to update the website with the new title and message.

In the **second situation**, we have to do more work because the event has been moved to a different day. This means that we have to remove the old event from the website, then make a clone of the new data, and finally, we update the website by adding the new event to the correct day.

1. Notice the code in **MAROON** uses the private variable that was created in chapter 1. The private variable was named **this.prevDay**.

2. The code in **MAROON** removes the **previous** event by using **this.prevDay** as the id

3. The code in **GREY** stores the updated date of the event into **this.day**, which represents the new date that the event will take place. Next, **this.day** is used as the id for making a clone

    This is how we are able to move an event from a day to a different day. We store the old date into **this.prevDay** and use it as an id. Next, JS finds it using `document.getElementById()` and then removes it from the website.

    We store the new date of the event into **this.day** and use it as an id to make a new clone. Finally, we add the new clone into a different row and that creates an effect of moving an event from a day to a different day.

4. Write The HTML Code In Between `<body> </body>.` Write the code in **BLUE** **and** **PURPLE** and position it under the code in **BLACK.**

```
<button
id      = "nEventB"
onclick = "make_obj('mar_events')"
>
Mar
</button>

<button
id      = "modEB"
onclick = "mod_eObj()"
>
      Mod
</button>
```

Explanation

Chapters 1 – 3 are JS code that defines the actions of moving an event from a day to a different day.

The code above connects HTML to JS by using a button with a signal. The signal is **onclick**. When we click on the word "**Mod**", HTML will send a signal to JS to call the function definition `"mod_eObj()"`

Run the code and test

1. Click on the button "**Jan**"
2. Move the mouse cursor over to the block that says "**testing title**".
    a. Click on it and the pop up banner will appear
3. Go up to the top and change the day from 1 → 3
4. Click on the word "**Mod**" and the event changes from day 1 to day 3

JS Challenge

Currently, our calendar has a field for the day that the event will take place. However, if we enter an alphabetical letter or a special character, our calendar still accepts it. We need to filter the input to only accept valid input.

What is the definition of "**valid input**"? It depends on the problem statement. For our calendar, valid input for "**day**" is any number between 1 and 31.

Write JS code to detect if the day **IS NOT** 1 or 31. If the day is an alphabetical or special character, then send an alert.