Implementing the Module Pattern for Reusable Code

Scenario:

In this activity, you will apply the **Module**, **Observer**, and **Singleton** patterns to build a structured JavaScript application. These patterns help you encapsulate code, manage communication between components, and maintain consistent state across the application.

Step 1: create a new html file

- Select File
- Select New File...
- Name file index.html
- Press enter
- Select OK

Step 2: build the html structure

Add the following code to the index.html file. Fill in the blanks to complete the code.

HTML:

```
<!DOCTYPE html>
<html lang="___">
<head>
  <meta charset="___">
  <meta name="viewport" content="___">
  <title>___</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <h1>___</h1>
  <div id="___"></div>
  <script src="main.js"></script>
</body>
</html>
```

Step 3: create a new JavaScript File

- Select File
- Select New File...
- Name file main.js
- Press enter

- Select OK

Step 4: implement the Module Pattern (main.js)

Implement the Module Pattern by completing the following code.

JavaScript:

```
const CalculatorModule = (function () {
  let result = ___;  // Initialize the result to 0

  function add(value) {
    result ___ value;
    displayResult();
  }

  function subtract(value) {
    result ___ value;
    displayResult();
  }

  function displayResult() {
    document.getElementById('___').textContent = `Result: ${___}`;  //
Update the UI
  }

  return {
    ___,  // Expose the add function
    ___   // Expose the subtract function
  };
})();
```

Step 5: implement the Observer Pattern (main.js)

Add the following code to define Subject and Observer classes.

JavaScript:

```
class Subject {
  constructor() {
    this.observers = ___;  // Initialize the observers list
  }

  subscribe(observer) {
    this.observers.___(observer);  // Add an observer
  }

  unsubscribe(observer) {
    this.observers = this.observers.filter(obs => obs !== ___);  //
Remove an observer
  }

  notify() {
    this.observers.forEach(observer => observer.___());  // Notify all
observers
```

```
  }
}

class Observer {
  constructor(name) {
    this.name = ___;  // Store the observer's name
  }

  update() {
    console.log(`${___} received notification!`);  // Log a notification
  }
}
```

Step 6: implement the Singleton Pattern (main.js)

Use the following template to create a Singleton class that manages application settings.

JavaScript:

```
class Settings {
  constructor() {
    if (Settings.___) {
      return ___;  // Return the existing instance
    }

    this.configuration = ___;  // Initialize the configuration object
    Settings.___ = this;  // Store the instance
  }

  set(key, value) {
    this.configuration[___] = ___;  // Set a configuration value
  }

  get(key) {
    return this.configuration[___];  // Retrieve a configuration value
  }
}
```

Step 7: run your code

- Click Go Live (in the lower right of the lab).
- A new tab should open up and display your webpage!
- If your code is not running as you expected, go to the next item to see the correct code.

HTML:

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Module, Observer, Singleton</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>

  <h1>Calculator</h1>
  <div id="result">Result: 0</div>
  <button onclick="CalculatorModule.add(5)">Add 5</button>
  <button onclick="CalculatorModule.subtract(2)">Subtract 2</button>

  <h2>Observer Demo</h2>
  <button onclick="runObserverDemo()">Run Observer Demo</button>
  <button onclick="clearObserverOutput()">Clear Observer Output</button>
  <div id="observer-output" class="output-box">
    <div class="placeholder">No observer output yet.</div>
  </div>

  <h2>Singleton Demo</h2>
  <button onclick="runSingletonDemo()">Run Singleton Demo</button>
  <button onclick="clearSingletonOutput()">Clear Singleton
Output</button>
  <div id="singleton-output" class="output-box">
    <div class="placeholder">No singleton output yet.</div>
  </div>

  <script src="main.js"></script>
</body>
</html>
```

CSS:

```css
body {
  font-family: Arial, sans-serif;
  margin: 40px;
  background-color: #f4f4f4;
  color: #333;
  line-height: 1.5;
}

h1, h2 {
  color: #333;
  margin-bottom: 15px;
}

h2 {
  margin-top: 40px;
}

#result {
  font-size: 24px;
  margin-bottom: 20px;
  padding: 12px 18px;
  background-color: #fff;
  border: 2px solid #ccc;
  border-radius: 6px;
  display: inline-block;
}

button {
  margin: 5px 10px 10px 0;
  padding: 10px 20px;
  font-size: 16px;
  cursor: pointer;
  border: none;
  border-radius: 6px;
  background-color: #3498db;
  color: white;
  transition: background-color 0.3s ease;
}

button:hover {
  background-color: #2980b9;
}

.output-box {
  margin-top: 10px;
  padding: 15px;
  background-color: #eef;
  border: 1px solid #bbb;
  border-radius: 6px;
  min-height: 40px;
  width: fit-content;
  max-width: 100%;
  box-shadow: 2px 2px 5px rgba(0, 0, 0, 0.05);
}

.observer-message {
  margin-bottom: 8px;
  padding: 8px 12px;
  background-color: #dff0d8;
  border-left: 4px solid #3c763d;
  border-radius: 4px;
  font-size: 15px;
  color: #2b542c;
}

input[type="text"] {
  padding: 8px;
  font-size: 16px;
  border: 1px solid #ccc;
  border-radius: 5px;
  margin-right: 10px;
}

input[type="text"]:focus {
  border-color: #3498db;
  outline: none;
}

.placeholder {
  color: #888;
  font-style: italic;
  font-size: 14px;
  padding: 4px;
}
```

JavaScript:

```javascript
// --- Helper Functions ---
function setPlaceholder(containerId, message) {
  const container = document.getElementById(containerId);
  if (container) {
    container.innerHTML = `<div class="placeholder">${message}</div>`;
    console.log(`Placeholder set in #${containerId}: "${message}"`);
  }
}

function clearPlaceholder(containerId) {
  const container = document.getElementById(containerId);
  if (container) {
    const placeholder = container.querySelector('.placeholder');
    if (placeholder) {
      placeholder.remove();
      console.log(`Placeholder cleared in #${containerId}`);
    }
  }
}

// --- CalculatorModule ---
window.CalculatorModule = (function () {
  let result = 0;

  function add(value) {
    result += value;
    console.log(`Added ${value}, new result: ${result}`);
    displayResult();
  }

  function subtract(value) {
    result -= value;
    console.log(`Subtracted ${value}, new result: ${result}`);
    displayResult();
  }

  function displayResult() {
    const resultElement = document.getElementById('result');
    if (resultElement) {
      resultElement.textContent = `Result: ${result}`;
    } else {
      console.warn("Element with ID 'result' not found.");
    }
  }

  return {
    add,
    subtract
  };
})();

// --- Observer Pattern ---
class Subject {
  constructor() {
    this.observers = [];
    console.log('Subject created.');
  }
  subscribe(observer) {
    this.observers.push(observer);
    console.log(`Subscribed: ${observer.name}`);
  }

  unsubscribe(observer) {
    this.observers = this.observers.filter(obs => obs !== observer);
    console.log(`Unsubscribed: ${observer.name}`);
  }

  notify() {
    console.log('Notifying observers...');
    this.observers.forEach(observer => observer.update());
  }
}

class Observer {
  constructor(name, outputElementId) {
    this.name = name;
    this.outputElementId = outputElementId;
    console.log(`Observer created: ${this.name}`);
  }

  update() {
    const output = document.getElementById(this.outputElementId);
    if (output) {
      const message = document.createElement('div');
      message.className = 'observer-message';
      message.textContent = `${this.name} received notification!`;
      output.appendChild(message);
      console.log(`${this.name} updated.`);
    } else {
      console.warn(`Element with ID '${this.outputElementId}' not
found.`);
    }
  }
}

window.runObserverDemo = function () {
  const output = document.getElementById('observer-output');
  if (output) output.innerHTML = '';
  clearPlaceholder('observer-output');

  const subject = new Subject();
  const observerA = new Observer("Observer A", "observer-output");
  const observerB = new Observer("Observer B", "observer-output");

  subject.subscribe(observerA);
  subject.subscribe(observerB);
  subject.notify();
};

window.clearObserverOutput = function () {
  setPlaceholder('observer-output', 'No observer output yet.');
};

// --- Singleton Pattern ---
class Settings {
  constructor() {
    if (Settings.instance) {
      console.log('Settings instance reused.');
      return Settings.instance;
    }

    console.log('Settings instance created.');
    this.configuration = {};
    Settings.instance = this;
  }

  set(key, value) {
    this.configuration[key] = value;
    console.log(`Setting set: ${key} = ${value}`);
  }

  get(key) {
    const value = this.configuration[key];
    console.log(`Setting get: ${key} = ${value}`);
    return value;
  }
}

window.runSingletonDemo = function () {
  clearPlaceholder('singleton-output');

  const settings1 = new Settings();
  settings1.set("theme", "dark");

  const settings2 = new Settings();

  const output = document.getElementById('singleton-output');
  if (output) {
    output.innerHTML = '';
    const message = document.createElement('div');
    message.textContent = `Theme from settings2:
${settings2.get("theme")}`;
    output.appendChild(message);
  } else {
    console.warn("Element with ID 'singleton-output' not found.");
  }
};

window.clearSingletonOutput = function () {
  setPlaceholder('singleton-output', 'No singleton output yet.');
};
```