Modal Window Project

> 🎓 **CS Perspective:** This project demonstrates the **state machine** concept—the modal has two states (visible/hidden), and user actions trigger state transitions. The pattern of showing/hiding via CSS classes rather than inline styles follows the **separation of concerns** principle. Using `querySelectorAll` returns a **NodeList** (array-like structure) requiring iteration. The keyboard event listener on `document` implements **global event handling**, and the condition check (`!modal.classList.contains('hidden')`) prevents unnecessary state transitions—a form of **guard condition** in state machine terminology. This is also an example of the **facade pattern**: `openModal()` and `closeModal()` provide simple interfaces to complex DOM manipulations.

📌 Selecting Multiple Elements

```
// Returns NodeList (array-like, must loop)
const btnsOpenModal = document.querySelectorAll('.show-modal');

// Loop through all buttons
for (let i = 0; i < btnsOpenModal.length; i++) {
  btnsOpenModal[i].addEventListener('click', openModal);
}
```

Method	Returns	Use Case
`querySelector()`	First match	Single element
`querySelectorAll()`	NodeList	Multiple elements

📌 Showing & Hiding with Classes

Toggle visibility by adding/removing CSS classes:

```
const openModal = function () {
  modal.classList.remove('hidden');
  overlay.classList.remove('hidden');
};

const closeModal = function () {
  modal.classList.add('hidden');
  overlay.classList.add('hidden');
};
```

CSS Hidden Class

```
.hidden {
  display: none;
}
```

Why Classes Over Inline Styles?

- ✅ Separation of concerns (styles in CSS)
- ✅ Can toggle multiple properties at once
- ✅ Easier to maintain and debug
- ✅ Better performance

📌 Multiple Event Listeners

Attach same handler to multiple elements:

```
for (let i = 0; i < btnsOpenModal.length; i++) {
  btnsOpenModal[i].addEventListener('click', openModal);
}
```

Note: Pass function **reference** (`openModal`), not function **call** (`openModal()`).

📌 Handling Keyboard Events

Listen for keyboard events on the entire document:

```
document.addEventListener('keydown', function (e) {
  console.log(e.key);  // Log which key was pressed

  if (e.key === 'Escape' && !modal.classList.contains('hidden')) {
    closeModal();
  }
});
```

Event Object Properties

Property	Description	Example
`e.key`	Key pressed	`'Escape'`, `'Enter'`
`e.code`	Physical key	`'KeyA'`, `'ArrowUp'`
`e.type`	Event type	`'keydown'`, `'keyup'`

Keyboard Event Types

Event	Fires When
`keydown`	Key is pressed (repeats if held)
`keyup`	Key is released
`keypress`	Character key pressed (deprecated)

📌 Guard Conditions

Check state before performing action:

```
// Only close if modal is currently visible
if (e.key === 'Escape' && !modal.classList.contains('hidden')) {
  closeModal();
}
```

This prevents calling `closeModal()` when the modal is already hidden.

📌 Overlay Pattern

The overlay creates a backdrop behind the modal:

```
<div class="overlay hidden"></div>
```

```
.overlay {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background-color: rgba(0, 0, 0, 0.5);
  z-index: 5;
}
```

Clicking the overlay also closes the modal:

```
overlay.addEventListener('click', closeModal);
```

📌 Code Organization

Extracting repeated logic into named functions:

```
// ❌ Repeated code
btnCloseModal.addEventListener('click', function () {
  modal.classList.add('hidden');
  overlay.classList.add('hidden');
});

// ✅ Reusable function
const closeModal = function () {
  modal.classList.add('hidden');
  overlay.classList.add('hidden');
};

btnCloseModal.addEventListener('click', closeModal);
overlay.addEventListener('click', closeModal);
```

Benefits

- **DRY** (Don't Repeat Yourself)
- Easier to maintain
- Single point of change
- Self-documenting code