```
// nice-classes.js - Corrected and Deconvoluted NICE Classes Manager

export default class NiceClassManager {
    constructor() {
        // This array ONLY stores classes the user has manually clicked.
        this.selectedClasses = [];
        this.classesData = {};
        // This property is controlled by the 'Enable Optional Coordination' checkbox.
        this.enableOptionalCoordination = true;
        this.onSelectionChange = null; // Callback for other modules
    }

    /**
     * Initializes the manager, connecting it to the UI toggles.
     */
    initialize() {
        this.setupCoordinationToggle();
    }

    /**
     * Fetches class data from the API and renders the grid.
     * This is called by the app-controller after a successful login.
     */
    async loadFromAPI() {
        try {
            const response = await fetch('/nice-classes');
            if (!response.ok) throw new Error(`Failed to load NICE classes: ${response.status}

            const data = await response.json();
            this.classesData = data.classes;

            this.renderClassesGrid();
            console.log('NICE classes loaded and rendered successfully.');

        } catch (error) {
            console.error('Failed to load NICE classes:', error);
        }
    }

    /**
     * Creates the grid of NICE class buttons in the UI.
     */
    renderClassesGrid() {
        const grid = document.getElementById('niceClassesGrid');
        if (!grid || !this.classesData) return;

        grid.innerHTML = '';

        Object.values(this.classesData).sort((a, b) => a.number - b.number).forEach(classData
            const button = this.createClassButton(classData.number, classData);
            grid.appendChild(button);
        });
        this.updateDisplay();
    }

    /**
     * Creates a single NICE class button with its tooltip (bubble).
     * This restores the hover-bubble functionality.
     */
    createClassButton(classId, classData) {
        const button = document.createElement('div');
        button.className = `nice-class-button-compact ${classData.type.toLowerCase()}`;
        button.dataset.classId = classId;
        const hasCoordination = classData.coordinated && classData.coordinated.length > 0;

        button.innerHTML = `
            ${classId}
            ${hasCoordination ? `<div class="coordinated-indicator-compact">${classData.coordi
            <div class="tooltip-compact">
                <strong>Class ${classId} (${classData.type.toUpperCase()})</strong><br>
                ${classData.description}
                ${hasCoordination ? `<br><em>Coordinates with: ${classData.coordinated.join(',
            </div>
        `;
        button.addEventListener('click', () => this.toggleClass(classId));
        return button;
    }

    /**
     * Handles a user clicking on a class button.
     * Its ONLY job is to add or remove the clicked class from the manual selection list.
     */
    toggleClass(classId) {
        const classIdNum = parseInt(classId);
        const index = this.selectedClasses.indexOf(classIdNum);

        if (index === -1) {
            this.selectedClasses.push(classIdNum);
        } else {
            this.selectedClasses.splice(index, 1);
        }
        // After changing the selection, update the entire grid's appearance.
        this.updateDisplay();
    }

    /**
     * This is the single source of truth for the final list of classes to be searched.
     * It performs all coordination logic based on the current state.
     */
    getSelectedClasses() {
        const finalSelection = new Set(this.selectedClasses);

        // Step 1: ALWAYS apply forced 9 <-> 42 coordination.
        if (finalSelection.has(9) || finalSelection.has(42)) {
            finalSelection.add(9);
            finalSelection.add(42);
        }

        // Step 2: Conditionally apply optional coordination ONLY if the toggle is enabled.
        if (this.enableOptionalCoordination) {
            // Iterate over a copy of the original manual selections to avoid infinite loops
            const manualSelections = [...this.selectedClasses];
            manualSelections.forEach(classId => {
                const classData = this.classesData[classId];
                if (classData && classData.coordinated) {
                    classData.coordinated.forEach(coordId => finalSelection.add(coordId));
                }
            });
        }

        return Array.from(finalSelection).sort((a, b) => a - b).map(String);
    }

    /**
     * Applies the "select all" visual state, called by search.js.
     */
    setSelectAllState(isSelectAll) {
        const buttons = document.querySelectorAll('.nice-class-button-compact');
        buttons.forEach(button => {
            button.classList.remove('selected', 'coordinated', 'select-all-active');
            if (isSelectAll) {
                button.classList.add('select-all-active');
            }
        });
        // If we are unchecking "select all", revert to the normal display.
        if (!isSelectAll) {
            this.updateDisplay();
        }
    }

    /**
     * Updates the visual state (CSS classes) of all buttons based on the final selection.
     */
    updateDisplay() {
        const buttons = document.querySelectorAll('.nice-class-button-compact');
        // Get the final, calculated list of ALL classes that should be active.
        const finalCoordinatedSet = new Set(this.getSelectedClasses().map(c => parseInt(c)));

        buttons.forEach(button => {
            const classId = parseInt(button.dataset.classId);
            // isManuallySelected is true only if the user clicked THIS specific button.
            const isManuallySelected = this.selectedClasses.includes(classId);
            // isCoordinated is true if it's in the final list BUT wasn't manually clicked.
            const isCoordinated = finalCoordinatedSet.has(classId) && !isManuallySelected;

            button.classList.toggle('selected', isManuallySelected);
            button.classList.toggle('coordinated', isCoordinated);
        });
    }

    /**
     * Connects the HTML checkbox to the enableOptionalCoordination property.
     */
    setupCoordinationToggle() {
        const toggle = document.getElementById('enableCoordination');
        if (toggle) {
            toggle.checked = this.enableOptionalCoordination;
            toggle.addEventListener('change', () => {
                this.enableOptionalCoordination = toggle.checked;
                // When the toggle changes, immediately update the display.
                this.updateDisplay();
            });
        }
    }

    /**
     * Clears the user's manual selections.
     */
    clearSelection() {
        this.selectedClasses = [];
        this.updateDisplay();
    }
}
```