Gestione elenco Persone

Esempio di un'applicazione JavaScript che utilizza i concetti di OOP in combinazione con HTML e CSS. Questa applicazione gestisce un elenco di **persone**, consentendo di aggiungere nuove persone e visualizzarle in un'interfaccia web.

Sommario

Struttura dei file

- **index.html**: Contiene la struttura HTML.
- **styles.css**: Contiene lo stile.
- **app.js**: Contiene la logica JavaScript e OOP.

Codice HTML (`index.html`)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Gestione Persone</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="container">
        <h1>Gestione Persone</h1>
        <form id="personaForm">
            <input type="text" id="nome" placeholder="Nome" required>
            <input type="number" id="eta" placeholder="Età" required>
            <button type="submit">Aggiungi Persona</button>
        </form>
        <ul id="listaPersone"></ul>
    </div>
    <script src="app.js"></script>
</body>
</html>
```

Codice CSS (`styles.css`)

```
body {
    font-family: Arial, sans-serif;
    background-color: #f9f9f9;
    margin: 0;
    padding: 0;
}

.container {
    max-width: 600px;
    margin: 50px auto;
    padding: 20px;
    background: #ffffff;
    border-radius: 8px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

h1 {
    text-align: center;
    color: #333;
}

form {
    display: flex;
    gap: 10px;
    margin-bottom: 20px;
}

form input, form button {
    padding: 10px;
    font-size: 16px;
}

form input {
    flex: 1;
    border: 1px solid #ccc;
    border-radius: 4px;
}

form button {
    background-color: #28a745;
    color: white;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    transition: background-color 0.3s;
}

form button:hover {
    background-color: #218838;
}

ul {
    list-style-type: none;
    padding: 0;
}

ul li {
    padding: 10px;
    margin-bottom: 5px;
    background: #f1f1f1;
```

```
    border: 1px solid #ddd;
    border-radius: 4px;
    display: flex;
    justify-content: space-between;
    align-items: center;
}

ul li button {
    background-color: #dc3545;
    color: white;
    border: none;
    border-radius: 4px;
    padding: 5px 10px;
    cursor: pointer;
    transition: background-color 0.3s;
}

ul li button:hover {
    background-color: #c82333;
}
```

Codice JavaScript (`app.js`)

```
// Classe Persona
class Persona {
    constructor(nome, eta) {
        this.nome = nome;
        this.eta = eta;
    }

    descrizione() {
        return `${this.nome}, ${this.eta} anni`;
    }
}

// Classe GestorePersone
class GestorePersone {
    constructor() {
        this.persone = []; // Array per memorizzare le persone
    }

    aggiungiPersona(persona) {
        this.persone.push(persona);
    }

    rimuoviPersona(index) {
        this.persone.splice(index, 1);
    }

    getListaPersone() {
        return this.persone;
    }
}

// Gestore dell'applicazione
const gestore = new GestorePersone();

// Riferimenti agli elementi DOM
const personaForm = document.getElementById('personaForm');
const nomeInput = document.getElementById('nome');
const etaInput = document.getElementById('eta');
const listaPersone = document.getElementById('listaPersone');

// Funzione per aggiornare la lista delle persone
function aggiornaLista() {
    listaPersone.innerHTML = '';
    gestore.getListaPersone().forEach((persona, index) => {
        const li = document.createElement('li');
        li.textContent = persona.descrizione();

        const rimuoviBtn = document.createElement('button');
        rimuoviBtn.textContent = 'Rimuovi';
        rimuoviBtn.onclick = () => {
            gestore.rimuoviPersona(index);
            aggiornaLista();
        };

        li.appendChild(rimuoviBtn);
        listaPersone.appendChild(li);
    });
}

// Gestione dell'evento submit del form
```

```
personaForm.addEventListener('submit', (e) => {
    e.preventDefault();

    const nome = nomeInput.value.trim();
    const eta = parseInt(etaInput.value.trim());

    if (nome && !isNaN(eta)) {
        const nuovaPersona = new Persona(nome, eta);
        gestore.aggiungiPersona(nuovaPersona);
        aggiornaLista();

        // Resetta il form
        nomeInput.value = '';
        etaInput.value = '';
    }
});
```

Come Funziona

1. **Concetti OOP Applicati**:
    - **Classe Persona**: Rappresenta un oggetto con proprietà (`nome` e `eta`) e metodo (`descrizione`).
    - **Classe GestorePersone**: Gestisce un array di oggetti `Persona` e fornisce metodi per aggiungere, rimuovere e ottenere persone.
    - **Incapsulamento**: I dettagli interni della gestione delle persone (array) sono nascosti dietro i metodi pubblici della classe `GestorePersone`.
2. **Interazione con il DOM**:
    - Il form consente di inserire i dati di una nuova persona.
    - Gli oggetti `Persona` vengono aggiunti alla lista gestita da `GestorePersone`.
    - L'interfaccia viene aggiornata dinamicamente tramite `aggiornaLista`.
3. **Stile CSS**:
    - Lo stile rende l'interfaccia più accattivante e leggibile, con effetti hover sui pulsanti.

Risultato

Quando apri il file `index.html` in un browser:

- Puoi aggiungere una persona inserendo il **nome** e l'**età** e cliccando su "Aggiungi Persona".
- L'elenco delle persone si aggiorna automaticamente.
- Puoi rimuovere una persona cliccando sul pulsante "Rimuovi" accanto al suo nome.

Questo progetto è un esempio completo di applicazione OOP in JavaScript con un'interfaccia utente!

Analisi Dettagliata dell'Esempio OOP in JavaScript

L'applicazione creata gestisce un elenco di persone, con un'interfaccia che consente di aggiungere e rimuovere individui dalla lista. Qui di seguito fornisco un'analisi **dettagliata riga per riga**, **specifiche funzionali**, **diagramma di flusso** e un'analisi **top-down** del progetto.

Codice JavaScript Dettagliato:

1. Definizione della Classe `Persona`

```
class Persona {
    constructor(nome, eta) { // Costruttore
        this.nome = nome; // Salva il nome
        this.eta = eta;   // Salva l'età
    }

    descrizione() { // Metodo per descrivere la persona
        return `${this.nome}, ${this.eta} anni`;
    }
}
```

- **`class Persona`**: Modella un oggetto con due proprietà (`nome`, `eta`) e un comportamento (`descrizione`).
- **`constructor(nome, eta)`**: Inizializza le proprietà con i valori passati.
- **`descrizione`**: Ritorna una stringa leggibile, utile per visualizzare i dati nell'interfaccia.

2. Definizione della Classe `GestorePersone`

```
class GestorePersone {
    constructor() {
        this.persone = []; // Array per memorizzare le persone
    }

    aggiungiPersona(persona) { // Aggiunge una persona all'elenco
        this.persone.push(persona);
    }

    rimuoviPersona(index) { // Rimuove una persona dato l'indice
        this.persone.splice(index, 1);
    }

    getListaPersone() { // Restituisce tutte le persone
        return this.persone;
    }
}
```

- **`class GestorePersone`**: Gestisce la logica dell'elenco di persone.
- **`aggiungiPersona(persona)`**: Aggiunge una nuova persona all'array `persone`.
- **`rimuoviPersona(index)`**: Rimuove la persona all'indice specificato.
- **`getListaPersone`**: Restituisce tutte le persone attualmente memorizzate.

3. Interazione con il DOM

```
const gestore = new GestorePersone(); // Istanzia il gestore delle persone
const personaForm = document.getElementById('personaForm'); // Form
const nomeInput = document.getElementById('nome'); // Input del nome
const etaInput = document.getElementById('eta'); // Input dell'età
const listaPersone = document.getElementById('listaPersone'); // Lista
visualizzata
```

- **`gestore`**: Oggetto che contiene la logica OOP per gestire le persone.
- Gli elementi HTML (`personaForm`, `nomeInput`, ecc.) vengono referenziati tramite `document.getElementById`, per poter interagire con essi dal codice JavaScript.

4. Aggiornamento della Lista

```
function aggiornaLista() {
    listaPersone.innerHTML = ''; // Svuota la lista
    gestore.getListaPersone().forEach((persona, index) => { // Itera sulle
persone
        const li = document.createElement('li'); // Crea un elemento della lista
        li.textContent = persona.descrizione(); // Assegna la descrizione

        const rimuoviBtn = document.createElement('button'); // Bottone
"Rimuovi"
        rimuoviBtn.textContent = 'Rimuovi';
        rimuoviBtn.onclick = () => { // Aggiunge l'evento di rimozione
            gestore.rimuoviPersona(index);
            aggiornaLista(); // Aggiorna la lista dopo la rimozione
        };

        li.appendChild(rimuoviBtn); // Aggiunge il bottone all'elemento
        listaPersone.appendChild(li); // Aggiunge l'elemento alla lista
    });
}
```

- **`aggiornaLista`**: Aggiorna il contenuto della lista in base all'array `persone`.
- Utilizza il metodo `getListaPersone` del gestore per ottenere l'elenco.
- Crea un pulsante di rimozione per ogni persona e aggiunge un evento `onclick`.

5. Gestione del Form

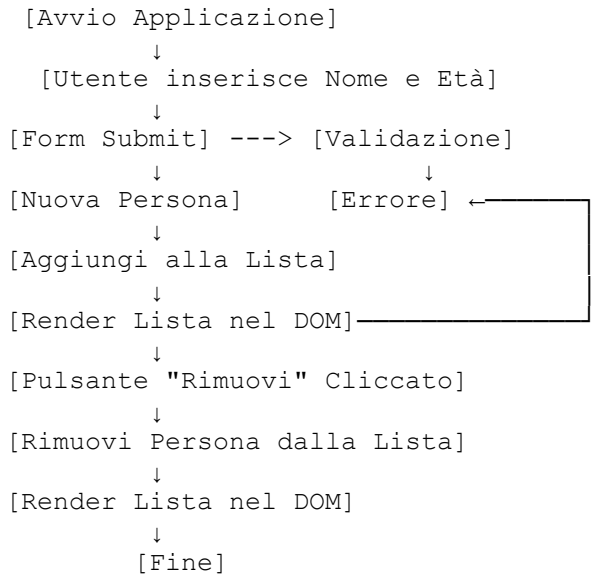
```
personaForm.addEventListener('submit', (e) => {
    e.preventDefault(); // Previene il comportamento di default del form

    const nome = nomeInput.value.trim(); // Ottiene e pulisce il nome
    const eta = parseInt(etaInput.value.trim()); // Ottiene e converte l'età

    if (nome && !isNaN(eta)) { // Controlla che i dati siano validi
        const nuovaPersona = new Persona(nome, eta); // Crea una nuova persona
        gestore.aggiungiPersona(nuovaPersona); // Aggiunge la persona al gestore
        aggiornaLista(); // Aggiorna la lista

        nomeInput.value = ''; // Resetta il form
        etaInput.value = '';
    }
});
```

- **`submit`**: Aggiunge un evento al form per gestire l'invio.
- Crea una nuova istanza di `Persona` solo se i dati sono validi.
- Chiama `aggiornaLista` per riflettere il cambiamento.

Specifiche Funzionali

1. **Aggiungere una persona**:
    - Inserire nome ed età.
    - La persona viene aggiunta alla lista e visualizzata nell'interfaccia.
2. **Rimuovere una persona**:
    - Premendo il pulsante "Rimuovi" accanto al nome, la persona viene eliminata.
3. **Aggiornare dinamicamente la lista**:
    - Ogni modifica si riflette immediatamente nell'interfaccia.

Diagramma di Flusso



Analisi Top-Down

1. **Obiettivo Generale**:
    - Creare un'applicazione per gestire un elenco di persone.
    - Offrire funzionalità per aggiungere, visualizzare e rimuovere persone.
2. **Divisione in Componenti**:
    - **Modello (Model)**:
        - **Classe Persona**: Rappresenta una persona con nome ed età.
        - **Classe GestorePersone**: Gestisce l'elenco delle persone.
    - **Vista (View)**:
        - Elementi HTML (`form`, `input`, `ul`) per interagire con l'utente.
    - **Controller**:
        - Eventi DOM e funzioni JavaScript per aggiornare il modello e la vista.
3. **Flusso dei Dati**:
    - L'utente inserisce i dati.
    - Il controller aggiunge una nuova persona al gestore.
    - Il gestore aggiorna l'elenco.
    - La vista riflette i cambiamenti.
4. **Iterazione tra Componenti**:
    - Le modifiche al modello (`GestorePersone`) si riflettono nella vista tramite la funzione `aggiornaLista`.

Conclusione

Questo esempio segue il paradigma OOP per separare **dati** (oggetti e classi) e **comportamenti** (metodi e funzioni). La struttura **modulare** permette di estendere facilmente l'applicazione, ad esempio aggiungendo funzionalità come la modifica dei dati.