Deber#2

Parcial I

Nombre: Mathius Steven Moyano Jara

NRC:13899

Manual paso a paso para crear una aplicación web en JavaScript implementando Programación Orientada a Objetos (OOP)

Índice

---

1. Introducción

La Programación Orientada a Objetos (OOP) es un paradigma de programación que utiliza "objetos" para modelar datos y comportamientos del mundo real en las aplicaciones de software. Este enfoque permite una mejor organización del código, mayor reutilización, y facilita el mantenimiento y la escalabilidad de las aplicaciones.

En el contexto del desarrollo web, JavaScript es un lenguaje versátil y ampliamente utilizado que soporta los principios de la OOP. Aunque históricamente JavaScript ha sido conocido principalmente por su uso en el desarrollo del frontend, la llegada de tecnologías como Node.js ha extendido su aplicabilidad al backend, permitiendo a los desarrolladores crear aplicaciones completas utilizando un solo lenguaje.

En esta tarea, el objetivo es crear una aplicación web que implemente los principios de la OOP en JavaScript. Para lograr esto, se diseñará un modelo de clases que ejemplifique los conceptos de generalización, asociación, agregación y composición. Este modelo se implementará en una aplicación web sencilla, que luego se desplegará en un servidor de hosting. A lo largo de este proceso, se documentarán todos los pasos en un manual en PDF y se generará un video explicativo para complementar la entrega.

Objetivos específicos:

1. **Aplicar los fundamentos de OOP en JavaScript**: Utilizar clases y objetos para estructurar el código de manera modular y eficiente.
2. **Desarrollar una aplicación web**: Crear una aplicación funcional que demuestre la implementación de los conceptos OOP.
3. **Documentar el proceso**: Crear un manual detallado y un video tutorial que expliquen cada paso del desarrollo y despliegue.
4. **Publicar la aplicación**: Asegurar que la aplicación esté accesible en un servidor web y compartir todos los recursos asociados en la plataforma virtual indicada.

Alcance:

La aplicación a desarrollar será sencilla pero debe cubrir los aspectos fundamentales de OOP. Se espera que el estudiante:

- Propone un modelo de clases adecuado para un caso del mundo real.
- Implemente este modelo en JavaScript utilizando los conceptos de generalización, asociación, agregación y composición.

- Integre la lógica de las clases en una interfaz web básica.
- Despliegue la aplicación en un servidor de hosting.
- Documente todo el proceso de manera exhaustiva.

Al finalizar esta tarea, el estudiante habrá adquirido una comprensión práctica de cómo aplicar la OOP en el desarrollo de aplicaciones web con JavaScript, lo cual es una habilidad valiosa en el ámbito profesional del desarrollo de software.

2. Requisitos

- Conocimientos básicos de JavaScript y HTML/CSS.

- Entorno de desarrollo: cualquier editor de código (Visual Studio Code recomendado).

- Navegador web para pruebas (Google Chrome, Firefox, etc.).

- Servidor web para el despliegue (sugerido por el tutor).

3. Configuración del entorno de desarrollo

1. Instalar Visual Studio Code (VSCode)

  - Descargar e instalar VSCode desde (https://code.visualstudio.com/).

2. Crear la estructura del proyecto:

  - Crear una carpeta para tu proyecto.

  - Dentro de la carpeta, crear los siguientes archivos:

    - `index.html`

    - `style.css`

    - `app.js`

4. Desarrollo del modelo de clases

Generalización

La generalización se refiere a la creación de una clase base (padre) que puede ser extendida por otras clases (hijas).

```javascript
// Clase base (padre)
class Persona {
  constructor(nombre, edad) {
    this.nombre = nombre;
    this.edad = edad;
  }

  saludar() {
    console.log(`Hola, mi nombre es ${this.nombre} y tengo ${this.edad} años.`);
  }
}

// Clase hija
class Estudiante extends Persona {
  constructor(nombre, edad, curso) {
    super(nombre, edad);
    this.curso = curso;
  }

  estudiar() {
    console.log(`${this.nombre} está estudiando el curso de ${this.curso}.`);
  }
}

// Ejemplo de uso
let estudiante1 = new Estudiante('Juan', 21, 'Matemáticas');
```

estudiante1.saludar(); // Hola, mi nombre es Juan y tengo 21 años.

estudiante1.estudiar(); // Juan está estudiando el curso de Matemáticas.

```

#### Asociación

La asociación implica una relación entre dos clases donde una clase utiliza o necesita la otra.

```javascript
class Profesor {
  constructor(nombre, materia) {
    this.nombre = nombre;
    this.materia = materia;
  }

  enseñar() {
    console.log(`${this.nombre} está enseñando ${this.materia}.`);
  }
}

class Aula {
  constructor(numero, profesor) {
    this.numero = numero;
    this.profesor = profesor;
  }

  iniciarClase() {
    console.log(`Clase en el aula ${this.numero} con ${this.profesor.nombre}.`);
```
```

```
    this.profesor.enseñar();
  }
}

// Ejemplo de uso
let profesor1 = new Profesor('Luis', 'Física');
let aula1 = new Aula(101, profesor1);
aula1.iniciarClase(); // Clase en el aula 101 con Luis. Luis está enseñando Física.
```

Agregación

La agregación representa una relación "tiene un" entre dos clases, donde una clase contiene una colección de objetos de otra clase.

```javascript
class Departamento {
  constructor(nombre) {
    this.nombre = nombre;
    this.profesores = [];
  }

  agregarProfesor(profesor) {
    this.profesores.push(profesor);
  }

  mostrarProfesores() {
    console.log(` Profesores en el departamento de ${this.nombre}:`);
    this.profesores.forEach(profesor => {
```

```
        console.log(` - ${profesor.nombre}`);
      });
    }
}

// Ejemplo de uso
let departamento1 = new Departamento('Ciencias');
departamento1.agregarProfesor(profesor1);
departamento1.mostrarProfesores(); // Profesores en el departamento de Ciencias: - Luis
```

Composición

La composición es una forma más fuerte de agregación donde la existencia de los objetos contenidos depende de la existencia del objeto contenedor.

```javascript
class Curso {
  constructor(nombre) {
    this.nombre = nombre;
    this.estudiantes = [];
  }

  agregarEstudiante(estudiante) {
    this.estudiantes.push(estudiante);
  }

  listarEstudiantes() {
```

```
    console.log(` Estudiantes en el curso de ${this.nombre}:` );
    this.estudiantes.forEach(estudiante => {
      console.log(` - ${estudiante.nombre}` );
    });
  }
}

// Ejemplo de uso
let curso1 = new Curso('Química');
curso1.agregarEstudiante(estudiante1);
curso1.listarEstudiantes(); // Estudiantes en el curso de Química: - Juan
```

5. Implementación de la aplicación web

En este paso, se integrarán los conceptos anteriores dentro de una aplicación web sencilla.

`index.html`

```html
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Aplicación OOP en JavaScript</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
```

```
    <div id="app">
        <h1>Aplicación OOP en JavaScript</h1>
        <!-- Contenido dinámico se agregará aquí -->
    </div>
    <script src="app.js"></script>
</body>
</html>
```

`style.css`

```css
body {
    font-family: Arial, sans-serif;
    margin: 20px;
    background-color: #f0f0f0;
}

#app {
    background-color: #fff;
    padding: 20px;
    border-radius: 5px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
```

`app.js`

```javascript
document.addEventListener('DOMContentLoaded', () => {
```

```
  // Crear instancias de las clases
  let profesor1 = new Profesor('Luis', 'Física');
  let estudiante1 = new Estudiante('Juan', 21, 'Matemáticas');
  let curso1 = new Curso('Química');
  curso1.agregarEstudiante(estudiante1);

  // Actualizar el DOM con el contenido dinámico
  let app = document.getElementById('app');
  app.innerHTML += `<p>${profesor1.nombre} está enseñando ${profesor1.materia}.</p>`;
  app.innerHTML += `<p>${estudiante1.nombre} está estudiando el curso de ${estudiante1.curso}.</p>`;
  app.innerHTML += `<p>Estudiantes en el curso de ${curso1.nombre}: ${curso1.estudiantes.map(e => e.nombre).join(', ')}</p>`;
});
```

6. Despliegue en servidor de hosting

1. **Seleccionar un servidor de hosting**: Utiliza el servidor sugerido por el tutor.
2. **Subir los archivos**: Sube todos los archivos del proyecto (`index.html`, `style.css`, `app.js`) al servidor.
3. **Probar la aplicación**: Accede a la URL proporcionada por el servidor para verificar que la aplicación funciona correctamente.

7. Creación del manual en PDF

1. **Documentar el proceso**: Utiliza un editor de texto como Microsoft Word o Google Docs para crear el manual.
2. **Incluir capturas de pantalla**: Añade capturas de pantalla de cada paso del desarrollo y despliegue.
3. **Exportar a PDF**: Guarda el documento en formato PDF.

8. Generación del video de explicación

1. **Grabar el video**: Utiliza una herramienta de grabación de pantalla (como OBS Studio) para grabar el proceso de desarrollo y despliegue.

2. **Editar el video**: Edita el video para agregar narraciones y aclaraciones necesarias.

3. **Exportar el video**: Guarda el video en un formato adecuado (como MP4).

9. Publicación de la aplicación

1. **Comprimir los archivos**: Crea un archivo `.rar` que contenga todos los archivos del proyecto, el manual en PDF y el video de explicación.

2. **Nombrar el archivo**: Nombra el archivo siguiendo el formato `Apellido_Nombre_Tarea2_js_poo.rar`.

3. **Subir el archivo**: Publica el archivo en la plataforma virtual indicada por el tutor.

---

10. Espacio para el código fuente

Aquí se puede adjuntar el código fuente completo del proyecto.

Con esto concluye el manual para la creación y despliegue de una aplicación web en JavaScript aplicando Programación Orientada a Objetos.