



# Universidad Autónoma de Querétaro

## Facultad de Informática

Licenciatura en informática  
Mireles Nieves Giovana Paola  
Expediente 325671  
11/05/2024



Crear una clase 'Persona':

1. Define una clase Persona con propiedades como nombre, edad y método para imprimir los detalles de la persona.

Crear subclases:

2. Extiende la clase Persona para crear subclases como Estudiante, Profesor, etc. Cada subclase debe tener sus propias propiedades y métodos además de las heredadas de la clase Persona.

```
JS Personajs JS Estudiantejs JS Profesorjs JS Main1js
taller practica > JS Personajs > ...
1
2 class Persona {
3   constructor(nombre, edad) {
4     this.nombre = nombre;
5     this.edad = edad;
6   }
7
8   imprimirDetalles() {
9     console.log(`Nombre: ${this.nombre}, Edad: ${this.edad}`);
10  }
11 }
12
13 module.exports = Persona;
```

```
JS Personajs JS Estudiantejs JS Profesorjs JS Main1js
taller practica > JS Estudiantejs > ...
1
2 const Persona = require('./Persona');
3
4 class Estudiante extends Persona {
5   constructor(nombre, edad, grado) {
6     super(nombre, edad);
7     this.grado = grado;
8   }
9
10  imprimirDetalles() {
11    super.imprimirDetalles();
12    console.log(`Grado: ${this.grado}`);
13  }
14
15  estudiar() {
16    console.log(`${this.nombre} está estudiando.`);
17  }
18 }
19
20 module.exports = Estudiante;
```

```
JS Personajs JS Estudiantejs JS Profesorjs JS Main1js
taller practica > JS Profesorjs > (e) Persona
1
2 const Persona = require('./Persona');
3
4 class Profesor extends Persona {
5   constructor(nombre, edad, asignatura) {
6     super(nombre, edad);
7     this.asignatura = asignatura;
8   }
9
10  imprimirDetalles() {
11    super.imprimirDetalles();
12    console.log(`Asignatura: ${this.asignatura}`);
13  }
14
15  enseñar() {
16    console.log(`${this.nombre} está enseñando ${this.asignatura}.`);
17  }
18 }
19
20 module.exports = Profesor;
```

```
JS Personajs JS Estudiantejs JS Profesorjs JS Main1js
taller practica > JS Main1js > ...
1
2 const Estudiante = require('./Estudiante');
3 const Profesor = require('./Profesor');
4
5 const main = () => {
6   const estudiante = new Estudiante('Ana', 20, 'Segundo Año');
7   const profesor = new Profesor('Jose Manuel', 45, 'Matemáticas');
8
9   console.log("Detalles del estudiante:");
10  estudiante.imprimirDetalles();
11  estudiante.estudiar();
12
13  console.log("\nDetalles del profesor:");
14  profesor.imprimirDetalles();
15  profesor.enseñar();
16 }
17
18 main();
19
```

Crear una clase 'Libro':

1. Crea una clase Libro que tenga propiedades como título, autor, y método para imprimir los detalles del libro.

Agregar métodos:

2. Agrega métodos a la clase Libro para prestar y devolver el



# Universidad Autónoma de Querétaro

## Facultad de Informática

Licenciatura en informática  
Mireles Nieves Giovana Paola  
Expediente 325671  
11/05/2024



libro, manteniendo un registro de quién tiene prestado el libro en un array.

Implementar herencia:

3. Crea una clase Empleado que herede de la clase Persona, y añada propiedades específicas de un empleado, como salario y cargo.

Utilizar getters y setters:

4. Modifica alguna de las propiedades de las clases para que utilicen getters y setters, por ejemplo, para validar datos de entrada.

```
JS Libro.js X JS persona2.js JS Empleado.js JS Main2.js
taller practica > JS Libro.js > ...
1
2 class Libro {
3   constructor(titulo, autor) {
4     this.titulo = titulo;
5     this.autor = autor;
6     this.prestadoA = null;
7     this.historialPrestamos = [];
8   }
9
10  imprimirDetalles() {
11    console.log(`Titulo: ${this.titulo}, Autor: ${this.autor}`);
12  }
13
14  prestar(nombre) {
15    if (this.prestadoA) {
16      console.log(`El libro ya está prestado a ${this.prestadoA}`);
17    } else {
18      this.prestadoA = nombre;
19      this.historialPrestamos.push(nombre);
20      console.log(`El libro ha sido prestado a ${nombre}`);
21    }
22  }
23
24  devolver() {
25    if (this.prestadoA) {
26      console.log(`El libro ha sido devuelto por ${this.prestadoA}`);
27      this.prestadoA = null;
28    } else {
29      console.log(`El libro no está prestado actualmente.`);
30    }
31  }
32
33  imprimirHistorialPrestamos() {
34    console.log(`Historial de préstamos:`, this.historialPrestamos.join(', '));
35  }
36
37  module.exports = Libro;
38  ...
39

JS Libro.js JS persona2.js X JS Empleado.js JS Main2.js
taller practica > JS persona2.js > ...
1
2 class Persona {
3   constructor(nombre, edad) {
4     this.nombre = nombre;
5     this.edad = edad;
6   }
7
8   imprimirDetalles() {
9     console.log(`Nombre: ${this.nombre}, Edad: ${this.edad}`);
10  }
11
12
13  module.exports = Persona;
14  ...
```



# Universidad Autónoma de Querétaro

## Facultad de Informática

Licenciatura en informática  
Mireles Nieves Giovana Paola  
Expediente 325671  
11/05/2024



```
JS Libro.js JS persona2.js JS Empleado.js X JS Main2.js
taller practica > JS Empleado.js > ...
1
2 const Persona = require('./persona');
3
4 class Empleado extends Persona {
5   constructor(nombre, edad, salario, cargo) {
6     super(nombre, edad);
7     this._salario = salario;
8     this.cargo = cargo;
9   }
10
11   get salario() {
12     return this._salario;
13   }
14
15   set salario(nuevoSalario) {
16     if (nuevoSalario > 0) {
17       this._salario = nuevoSalario;
18     } else {
19       console.log('El salario debe ser un valor positivo.');

```
JS Libro.js JS persona2.js JS Empleado.js JS Main2.js X
taller practica > JS Main2.js > ...
1
2 const Libro = require('./Libro');
3 const Empleado = require('./Empleado');
4
5 const main = () => {
6   // Crear y probar la clase Libro
7   const libro = new Libro('Cien Años de Soledad', 'Gabriel García Márquez');
8   libro.imprimirDetalles();
9   libro.prestar('Juan');
10  libro.imprimirHistorialPrestamos();
11  libro.devolver();
12  libro.imprimirHistorialPrestamos();
13
14  // Crear y probar la clase Empleado
15  const empleado = new Empleado('Ana', 30, 50000, 'Ingeniera');
16  empleado.imprimirDetalles();
17  empleado.salario = 60000; // Usar setter para actualizar el salario
18  empleado.imprimirDetalles();
19  empleado.salario = -5000; // Intentar establecer un salario negativo
20 }
21
22 main();
23
```


```

Crear una clase 'Rectángulo':

1. Define una clase Rectángulo con propiedades como ancho y alto, y métodos para calcular el área y el perímetro del rectángulo.

Crear una clase 'Círculo':

2. Crea una clase Círculo con propiedades como radio y métodos para calcular el área y la circunferencia del círculo.

Crear una clase 'FiguraGeometrica':

3. Define una clase abstracta FiguraGeometrica con métodos abstractos como calcularArea y calcularPerimetro, y luego implementa subclases como Rectángulo y Círculo que hereden de esta clase y proporcionan implementaciones concretas de esos métodos.

Aplicar polimorfismo:

4. Crea una función que tome un objeto de tipo FiguraGeometrica como parámetro y utilice sus métodos calcularArea y calcularPerimetro sin importar si el objeto es un rectángulo o un círculo.

5.



# Universidad Autónoma de Querétaro

## Facultad de Informática

Licenciatura en informática  
Mireles Nieves Giovana Paola  
Expediente 325671  
11/05/2024



```
JS FigurasGeometricas.js X JS Rectangulo.js JS Circulo.js JS Main.js
taller practica > JS FigurasGeometricas.js > ...
1
2 class FiguraGeometrica {
3   constructor() {
4     if (this.constructor === FiguraGeometrica) {
5       throw new Error("No se puede instanciar una clase abstracta.");
6     }
7   }
8
9   calcularArea() {
10    throw new Error("Método abstracto 'calcularArea' no implementado.");
11  }
12
13   calcularPerimetro() {
14    throw new Error("Método abstracto 'calcularPerimetro' no implementado.");
15  }
16 }
17
18 module.exports = FiguraGeometrica;
```

```
JS FigurasGeometricas.js JS Rectangulo.js X JS Circulo.js JS Main.js
taller practica > JS Rectangulo.js > ...
1
2 module.exports = Rectangulo;
3
4
5 const FiguraGeometrica = require('./figuraGeometrica');
6
7 class Rectangulo extends FiguraGeometrica {
8   constructor(anch, alto) {
9     super();
10    this.anch = anch;
11    this.alto = alto;
12  }
13
14   calcularArea() {
15    return this.anch * this.alto;
16  }
17
18   calcularPerimetro() {
19    return 2 * (this.anch + this.alto);
20  }
21 }
22
23 module.exports = Rectangulo;
```

```
JS FigurasGeometricas.js X JS Rectangulo.js JS Circulo.js X JS Main3.js
taller practica > JS Circulo.js > ...
1
2 const FiguraGeometrica = require('./figuraGeometrica');
3
4 class Circulo extends FiguraGeometrica {
5   constructor(radio) {
6     super();
7     this.radio = radio;
8   }
9
10   calcularArea() {
11    return Math.PI * this.radio * this.radio;
12  }
13
14   calcularPerimetro() {
15    return 2 * Math.PI * this.radio;
16  }
17 }
18
19 module.exports = Circulo;
20
```

```
JS FigurasGeometricas.js JS Rectangulo.js JS Circulo.js JS Main3.js X
taller practica > JS Main3.js > ...
1
2 const Rectangulo = require('./Rectangulo');
3 const Circulo = require('./Circulo');
4
5 const mostrarDetalles = (figura) => {
6   console.log(`Área: ${figura.calcularArea()}`);
7   console.log(`Perímetro: ${figura.calcularPerimetro()}`);
8 };
9
10 const main3 = () => {
11   const rectangulo = new Rectangulo(10, 5);
12   const circulo = new Circulo(7);
13
14   console.log('Detalles del rectángulo:');
15   mostrarDetalles(rectangulo);
16
17   console.log('\nDetalles del círculo:');
18   mostrarDetalles(circulo);
19 };
20
21 main3();
```



# Universidad Autónoma de Querétaro

## Facultad de Informática

Licenciatura en informática  
Mireles Nieves Giovana Paola  
Expediente 325671  
11/05/2024



Analiza el siguiente código en JavaScript y determina qué patrón de diseño se está utilizando y describirlos

<pre>class Database {   constructor() {     if (!Database.instance) {       Database.instance = this;     }     return Database.instance;   }    query(sql) {     console.log("Ejecutando consulta:", sql);   } }  const db1 = new Database(); const db2 = new Database();  console.log(db1 === db2); // Output: true db1.query("SELECT * FROM users");</pre>	<p>Utiliza el <b>Patrón de Diseño Singleton</b>, este patrón se asegura de que una clase tenga solo una instancia y proporciona un punto de acceso global a ella.</p> <p><b>Constructor:</b> En el constructor de la clase <code>Database</code>, se verifica si <code>Database.instance</code> ya existe, si no existe, se asigna la instancia actual (<code>this</code>) a <code>Database.instance</code>, luego, el constructor retorna <code>Database.instance</code>.</p> <p><b>Verificación de la instancia:</b> Al crear dos instancias <code>db1</code> y <code>db2</code> de <code>Database</code>, ambas variables referencian la misma instancia debido a la lógica en el constructor. Esto se verifica con <code>console.log(db1 === db2);</code> que imprimirá <code>true</code>.</p> <p><b>Método query:</b> El método <code>query</code> es un método típico de la clase <code>Database</code> que simplemente imprime la consulta <code>SQL</code> que se le pasa como argumento.</p>
<pre>1 class Logger { 2   constructor() { 3     this.logs = []; 4   } 5 6   log(message) { 7     this.logs.push(message); 8     console.log("Log registrado:", message); 9   } 10 11   static getInstance() { 12     if (!Logger.instance) { 13       Logger.instance = new Logger(); 14     } 15     return Logger.instance; 16   } 17 } 18 19 const logger1 = Logger.getInstance(); 20 const logger2 = Logger.getInstance(); 21 22 console.log(logger1 === logger2); // Output: true 23 logger1.log("Error: No se puede conectar al servidor");</pre>	<p>Utiliza el <b>Patrón de Diseño Singleton</b> en este caso la clase <code>Logger</code> tiene un método estático <code>getInstance()</code> que devuelve siempre la misma instancia de la clase <code>Logger</code>.</p> <p><b>Constructor:</b> El constructor de la clase <code>Logger</code> inicializa una propiedad <code>logs</code> como un <code>array vacío</code>.</p> <p><b>Método log:</b> Este método toma un mensaje como argumento, lo agrega al array <code>logs</code> y luego lo imprime en la consola.</p> <p><b>Método estático getInstance():</b> Si no existe una instancia previa de</p>



# Universidad Autónoma de Querétaro

## Facultad de Informática

Licenciatura en informática  
Mireles Nieves Giovana Paola  
Expediente 325671  
11/05/2024



	<p><b>Logger</b>, crea una nueva instancia y la asigna a <b>Logger.instance</b>. Si ya existe una instancia, simplemente devuelve la instancia existente por lo cual esto garantiza que solo haya una instancia de <b>Logger</b> en toda la aplicación.</p> <p><b>Uso del Singleton:</b> Al llamar a <b>Logger.getInstance()</b>, se obtiene la misma instancia de <b>Logger</b> en todos los lugares donde se llama, como se puede ver en la comparación <b>logger1 === logger2</b>, que devuelve true. Esto confirma que <b>logger1</b> y <b>logger2</b> se refieren a la misma instancia de <b>Logger</b>.</p> <p><b>Llamada al método log:</b> Se llama al <b>método log</b> en <b>logger1</b>, agregando un mensaje al <b>array logs</b> y escribiendo el mensaje en la consola.</p>
<pre>1 class User { 2   constructor(name) { 3     this.name = name; 4   } 5 6   greet() { 7     console.log("Hola, soy", this.name); 8   } 9 } 10 11 class UserFactory { 12   createUser(name) { 13     return new User(name); 14   } 15 } 16 17 const factory = new UserFactory(); 18 const user1 = factory.createUser("Juan"); 19 const user2 = factory.createUser("Maria"); 20 21 user1.greet(); // Output: Hola, soy Juan 22 user2.greet(); // Output: Hola, soy Maria</pre>	<p>Utiliza el <b>Patrón de Diseño Factory Method</b> que define una interfaz para crear objetos en una superclase, pero permite que las subclases alteren el tipo de objetos que se crearán.</p> <p><b>Clase User:</b> Tiene un constructor que toma un name como argumento y lo asigna a la propiedad name.</p> <p><b>Tiene un método greet()</b> que imprime un saludo en la consola usando el nombre del usuario.</p> <p><b>Clase UserFactory:</b> Tiene un método <b>createUser(name)</b> que toma un name como argumento y devuelve una <b>nueva instancia de la clase User</b>.</p> <p><b>Uso del Factory Method:</b> Se crea una instancia de <b>UserFactory</b> llamada <b>factory</b>, se utilizan los métodos <b>factory.createUser("Juan")</b> y <b>factory.createUser("Maria")</b> para</p>



# Universidad Autónoma de Querétaro

## Facultad de Informática

Licenciatura en informática  
Mireles Nieves Giovana Paola  
Expediente 325671  
11/05/2024



	<p>crear dos instancias de User, <code>user1</code> y <code>user2</code>, respectivamente.</p> <p><b>Invocación del método <code>greet()</code>:</b></p> <p>Se llama al <code>método greet()</code> en <code>user1</code>, que <code>imprime "Hola, soy Juan"</code>.</p> <p>Se llama al <code>método greet()</code> en <code>user2</code>, que <code>imprime "Hola, soy Maria"</code>.</p>
--	---