


Herencia - Ejercicio practico

Ejercicios prácticos

js

 Copiar código

```
class Bus {
  constructor(nPasajero, pasajes) {
    this.nPasajero = nPasajero;
    this.pasajes = pasajes;
  }

  static precioPasaje = 20000;

  mensaje() {
    let totalP = this.pasajes * Bus.precioPasaje;
    console.log(`Bienvenido ${this.nPasajero}, has comprado ${this.pasajes} pasajes y el total a`
  }
}


class Bus1 extends Bus {
  constructor(nPasajero, pasajes, destino) {
    super(nPasajero, pasajes);
    this.destino = destino;
  }

  mensaje() {
    super.mensaje();
    console.log(`Nuestra ruta es ${this.destino}`);
    if (this.pasajes >= 4) {
      let descuento = this.pasajes * Bus.precioPasaje * 0.15;
      let totalP = this.pasajes * Bus.precioPasaje - descuento;
      console.log(`Por tu compra tienes un descuento del 15%, es decir solo pagas ${totalP}`);
    }
  }
}

const bus1 = new Bus1("Juan", 2, "Ibague");
console.log(bus1);
bus1.mensaje();
```

Ejercicio

js

 Copiar código

```
// Creamos una clase principal
class Empleado{
  constructor(nombre, edad, genero){
    this.nombre = nombre;
    this.edad = edad;
    this.genero = genero;
  }
  // Con el método 'static' podemos usar esta función sin necesidad de instanciar la clase.
  static cuentaCobro(){
    console.log("Pasar la cuenta de cobro al llegar al día '3' de casa mes")
  }
}

// Llamado a la función sin instanciar la clase.
Empleado.cuentaCobro();
```

```
// Usamos herencia
class Profesor extends Empleado{

  constructor(nombre, edad, genero, asignatura){
    super(nombre, edad, genero);
    this.asignatura = asignatura;
    this.horas = null;
  }
  // para poder acceder a 'horas' que no esta definido como atributo usamos 'get'. Me permite ver
  get getHoras(){
    return this.horas;
  }
  // Para poder tomar el valor usamos 'set'
  set setHoras(horas){
    this.horas = horas;
  }
}


const empleado = new Empleado("Luis", 23, "Masculino");
console.log(empleado);

const docente = new Profesor("Raul", 33, "Masculino","Español");
console.log(docente);
console.log(docente.getHoras);

// Para usar 'set' y poder darle un valor, hacemos lo siguiente:
docente.setHoras = 45;
console.log(docente.getHoras);
```

Herencia ejercicio practico 2

js

 Copiar código

```
class Animal {
  constructor(nombre, tipo) {
    this.nombre = nombre;
    this.tipo = tipo;
  }

  animalMessa() {
    console.log(`Hola, soy un ${this.tipo} y me llamo ${this.nombre} y sí, también hablo 😊`);
  }
}

class Animal1 extends Animal {
  constructor(nombre, tipo, patas = null) {
    super(nombre, tipo);
    this.patas = patas;
  }

  getPatas() {
    return this.patas;
  }

  setPatas(patas) {
    this.patas = patas;
  }
}

const ballena = new Animal1("Moby Dick", "Mamífero");
console.log(ballena);
ballena.setPatas(2); // Utiliza el método setPatas para establecer el número de patas
console.log(ballena.getPatas()); // Utiliza el método getPatas para obtener el número de patas

console.log(ballena);
```


Poliformismo

Permite que objetos de diferentes clases respondan al mismo mensaje de maneras distintas. Esto significa que un método puede tener diferentes implementaciones según la clase del objeto que lo esté utilizando, lo que brinda flexibilidad en el diseño del código y promueve la reutilización de código.

Sobreescritura de métodos

Permite que una clase hija reemplace la implementación de un método heredado de su clase padre con su propia implementación. Esto significa que la clase hija puede personalizar el comportamiento de un método heredado según sus necesidades específicas.

js

 Copiar código

```
// Sobreescritura de métodos


class Personas{
  constructor(nombre, edad){
    this.nombre = nombre;
    this.edad = edad;
  }
  saludar(){
    console.log(`Buenas noches, mi nombre es ${this.nombre} y tengo ${this.edad} años.`);
  }
}

class Desarrollador extends Personas{
  constructor(nombre, edad, tipo){
    super(nombre, edad);
    this.tipo = tipo;
  }
  // Sobreescribimos el metodo 'saludar()'
  saludar(){
    console.log(`Soy un desarrollador ${this.tipo}.`)
  }
}

let persona = new Personas("Carlos", 22);
console.log(persona);
persona.saludar();

let desarrollador = new Desarrollador("Bryan", 33, "FullStack");
console.log(desarrollador);
desarrollador.saludar();
```

js

 Copiar código

```
// Polimorfismo
class Personas{
  constructor(nombre, edad){
    this.nombre = nombre;
    this.edad = edad;
  }
  saludar(){
    console.log(`Buenas noches, mi nombre es ${this.nombre} y tengo ${this.edad} años.`);
  }
}
```

```

}

class Desarrollador extends Personas{
  constructor(nombre, edad, tipo){
    super(nombre, edad);
    this.tipo = tipo;
  }
  // Usando Polimorfismo quedaría de la siguiente forma:
  saludar(){
    super.saludar();
    console.log(`Soy un desarrollador ${this.tipo}.`);
  }
}

let persona = new Personas("Carlos", 22);
console.log(persona);
persona.saludar();

let desarrollador = new Desarrollador("Bryan", 33, "FullStack");
console.log(desarrollador);
desarrollador.saludar();

```

js

✓ Copiado!

```

// Ejercicio práctico
class Bus {
  constructor(nPasajero, pasajes) {
    this.nPasajero = nPasajero;
    this.pasajes = pasajes;
  }

  static precioPasaje = 20000;

  mensaje() {
    let totalP = this.pasajes * Bus.precioPasaje;
    console.log(`Bienvenido ${this.nPasajero}, has comprado ${this.pasajes} pasajes y el total a
  }
}

class Bus1 extends Bus {
  constructor(nPasajero, pasajes, destino) {
    super(nPasajero, pasajes);
    this.destino = destino;
  }

  mensaje() {
    super.mensaje();
    console.log(`Nuestra ruta es ${this.destino}`);
    if (this.pasajes >= 4) {
      let descuento = this.pasajes * Bus.precioPasaje * 0.15;
      let totalP = this.pasajes * Bus.precioPasaje - descuento;
      console.log(`Por tu compra tienes un descuento del 15%, es decir solo pagas ${totalP}`);
    }
  }
}

const bus1 = new Bus1("Juan", 2, "Ibague");
console.log(bus1);
bus1.mensaje();

```