

PROYECTO CLASES CON HERENCIA




Tabla de contenido

1. Programación orientada a objetos en typeScript. ...	3
2. Herencia aplicada al proyecto.	3
3. Sobre escritura de los métodos	5
4. Uso de la encapsulación en las diferentes clases: .	6
5. Polimorfismo	7

1. Programación orientada a objetos en TypeScript.

La programación orientada a objetos (Object Oriented Programming, OOP) es un modelo de programación informática que organiza el diseño de software en torno a datos u objetos, en lugar de funciones y lógica. Un objeto se puede definir como un campo de datos que tiene atributos y comportamiento únicos.

En el caso de nuestro proyecto los objetos estaban referidos a tipos de vehículos los cuales heredaban ciertas características los unos de los otros, en nuestro caso lo hemos planteado de tal manera que creamos una clase genérica de vehículo la cual establece unas premisas como el precio base o el tamaño de la rueda, dicho tamaño de la rueda será heredado por los otros dos objetos con los que tratamos en este proyecto, patines y bicicletas.

```
export class Vehículo {  
  protected _precioBase: number; // acceso subclase  
  private _tamañoRueda: number;  
  constructor(precioBase: number, tamañoRueda: number) {  
    this._precioBase = precioBase;  
    this._tamañoRueda = tamañoRueda;  
  }  
  get precioBase() {  
    return this._precioBase;  
  }  
  get tamañoRueda() {  
    return this._tamañoRueda;  
  }  
  precio(): number {  
    let precio: number;  
    precio = this._precioBase;   
    if (this._tamañoRueda > 26) {  
      precio += 0.15 * precio;  
    }  
  }  
}
```

Como podemos ver el precio base de todos los vehículos variará en función del tamaño de la rueda, además de por otros atributos únicos de cada vehículo.

2. Herencia aplicada al proyecto.

La herencia es uno de los puntos más importantes de este proyecto ya que de ella depende la fluctuación de los precios de los diferentes vehículos.

Como hemos comentado anteriormente el campo tamañoRueda se hereda entre las diferentes clases de este proyecto ya que el precio de cada vehículo depende de dicho campo:

```
precio(): number {  
  let precio: number;  
  precio = this._precioBase;  
  if (this._tamañoRueda > 26) {  
    precio += 0.15 * precio;  
  }  
}
```

```
import { Vehiculo } from './vehiculos';  
export class Bicicleta extends Vehiculo {  
  private _freno: string;  
  constructor(precioBase: number, tamañoRueda: number, freno: string) {  
    super(precioBase, tamañoRueda); //constructor de la superclase  
    this._freno = freno;  
  }  
  get freno() {  
    return this._freno;  
  }  
  get prueba(){  
    return this._precioBase // accedo si es protected  
  }  
  // sobre escribimos método  
  precio(): number {  
    let precio: number;  
    precio = super.precio();  
    if (this._freno == 'disco') {  
      precio += 0.15 * precio;  
    }  
  }  
}
```

Como podemos ver en la imagen la clase Bicicleta hereda campos de la clase vehículo.

```
import { Vehiculo } from './vehiculos';  
export class Patin extends Vehiculo {  
  private _rodamientos: string;  
  constructor(precioBase: number, tamañoRueda: number, rodamientos: string) {  
    super(precioBase, tamañoRueda); // constructor de la superclase  
    this._rodamientos = rodamientos;  
  }  
  get rodamientos() {  
    return this._rodamientos;  
  }  
  get prueba(){  
    return this._precioBase // accedo si es protected  
  }  
  // sobre escribimos método  
  precio(): number {  
    let precio: number;  
    precio = super.precio();  
    if (this._rodamientos == 'tipo A') {  
      precio += 0.35 * precio;  
    }  
    return precio;  
  }  
}
```

En el caso de los patines también ocurre lo mismo ya que es una clase que se extiende de la super clase

Toda la herencia que estamos viendo ocurre gracias a que dentro de las subclases extraemos la información de la superclase mediante el siguiente método:

```
constructor(precioBase: number, tamañoRueda: number, rodamientos: string) {  
  super(precioBase, tamañoRueda); // constructor de la superclase  
  this._rodamientos = rodamientos;  
}
```

3. Sobre escritura de los métodos

En nuestro proyecto hemos utilizado sobre escritura de los métodos ya que nos es primordial a la hora de calcular los precios de los diferentes vehículos.

En la siguiente imagen podemos observar cual sería el método original de un vehículo base:

```
precio(): number {  
  let precio: number;  
  precio = this._precioBase;  
  if (this._tamañoRueda > 26) {  
    precio += 0.15 * precio;  
  }  
}
```

Como podemos ver esta es la fórmula primitiva de cálculo de precio, aunque cuando estamos hablando de bicis además de tener en cuenta el tamaño de las ruedas también hemos de tener en cuenta el tipo de freno del que disponen por lo que para conseguirlo sobre escribimos el método precio en la subclase bicicleta:

```
precio(): number {  
  let precio: number;  
  precio = super.precio();  
  if (this._freno == 'disco') {  
    precio += 0.15 * precio;  
  }  
  return precio;  
}
```

Lo mismo ocurriría con los patines solo que en este caso se tendría en cuenta el tipo de rodamiento que utilizan:

```
precio(): number {  
  let precio: number;  
  precio = super.precio();  
  if (this._rodamientos == 'tipo A') {  
    precio += 0.35 * precio;  
  }  
  return precio;  
}
```

4. Uso de la encapsulación en las diferentes clases:

En nuestro proyecto estamos trabajando con campos que no queremos que sean modificados por el usuario final, es por ello que para evitar que el usuario final trabaje de forma directa con dichos campos hemos procedido a encapsularlos.

El resultado de la encapsulación en las diferentes clases es el siguiente:

```
export class Vehiculo {  
  protected _precioBase: number; // acceso subclase  
  private _tamañoRueda: number;  
  constructor(precioBase: number, tamañoRueda: number) {  
    this._precioBase = precioBase;  
    this._tamañoRueda = tamañoRueda;  
  }  
  get precioBase() {  
    return this._precioBase;  
  }  
  get tamañoRueda() {  
    return this._tamañoRueda;  
  }  
}
```

Como podemos ver el caso de las subclases únicamente encapsulamos los campos propios de dicha clase ya que el resto viene encapsulado en la super clase:

```
export class Bicicleta extends Vehiculo {  
  private _freno: string;  
  constructor(precioBase: number, tamañoRueda: number, freno: string) {  
    super(precioBase, tamañoRueda); //constructor de la superclase  
    this._freno = freno;  
  }  
  get freno() {  
    return this._freno;  
  }  
}
```

```
export class Patin extends Vehiculo {  
  private _rodamientos: string;  
  constructor(precioBase: number, tamañoRueda: number, rodamientos: string) {  
    super(precioBase, tamañoRueda); // constructor de la superclase  
    this._rodamientos = rodamientos;  
  }  
  get rodamientos() {  
    return this._rodamientos;  
  }  
}
```

5. Polimorfismo

En nuestro proyecto también hemos hecho uso del polimorfismo ya que hemos almacenado los diferentes objetos creados en un array por lo tanto con el fin de listar el contenido del ARRAY es imprescindible el uso de un bucle for el cual cuente con una variable la cual se transforme en función del valor que esté tomando.

A continuación podemos ver el array que hemos construido:

```
automoviles[0] = new Vehiculo(100, 28);
automoviles[1] = new Vehiculo(350, 10);
automoviles[2] = new Bicicleta(400, 29, 'disco');
automoviles[3] = new Bicicleta(150, 25, 'disco');
automoviles[4] = new Patin(90, 9, 'tipo B');
automoviles[5] = new Patin(100, 12.5, 'tipo A');
```

Como podemos ver el array cuenta con objetos de diferente tipo, tipo vehículo, bicicleta y patín.

Con el fin de listar este contenido ejecutamos un bucle for como el siguiente:

```
for (let b of automoviles) {
  console.log(b instanceof Bicicleta);
  console.log(`${b.todo()}, precio: ${b.precio()}`)
}
```

Este bucle nos dará como resultado todos los objetos del array y marcará true cuando recorra el objeto que sea de tipo Bicicleta.

La variable b es la que ejecuta el polimorfismo y la que va tomando diferentes formas en función del objeto que este recorriendo.

Dicho bucle arroja lo siguiente:

```
LISTAR CONTENIDO ARRAY
false
Precio base: 100, tamaño: 28, precio: 115
false
Precio base: 350, tamaño: 10, precio: 350
true
Precio base: 400, tamaño: 29, Tipo de freno: disco, precio: 529
true
Precio base: 150, tamaño: 25, Tipo de freno: disco, precio: 172.5
false
Precio base: 90, tamaño: 9, Tipo de rodamiento: tipo B, precio: 90
false
Precio base: 100, tamaño: 12.5, Tipo de rodamiento: tipo A, precio: 135
```