

Modelado de un objeto Producto

Este código define un constructor `Product` que modela un producto con varias propiedades (como `serial`, `brand`, `model`, `price`, etc.) y establece ciertos mecanismos para validar, acceder y modificar estas propiedades a través de getters y setters. También incluye cálculos automáticos relacionados con impuestos (IVA) y funcionalidades adicionales como la obtención de la descripción del producto y su precio sin impuestos.

Voy a desglosar el código paso a paso:

1. Validación del uso de new

```
function Product(serial, brand, model, price,  
    taxPercentage = Product.IVA) {  
  
if (!(this instanceof Product))  
    throw new InvalidAccessConstructorException();
```

- Esta línea verifica si `Product` se ha llamado como un constructor (con `new`). Si no, lanza una excepción **`InvalidAccessConstructorException`**. Esto garantiza que no se pueda instanciar **`Product`** como una función sin `new`.

2. Validación de parámetros requeridos

```
if (!serial) throw new EmptyValueException("serial");  
if (!brand) throw new EmptyValueException("brand");  
if (!model) throw new EmptyValueException("model");
```

- Aquí se validan los parámetros obligatorios (`serial`, `brand` y `model`). Si alguno de estos no se proporciona o está vacío, lanza una excepción `EmptyValueException`, indicando cuál valor está vacío.

3. Validación de price y taxPercentage

```
price = Number.parseFloat(price);  
if (!price || price <= 0) throw new InvalidValueException("price", price);  
if (!taxPercentage || taxPercentage < 0) throw new  
InvalidValueException("taxPercentage", taxPercentage);
```

- `price` se convierte a un número flotante (decimal). Si el precio no es un número o es menor o igual a 0, se lanza una excepción **`InvalidValueException`**.

- `taxPercentage` también se valida. Si no se proporciona o es menor a 0, se lanza otra excepción **`InvalidValueException`**.

4. Propiedades privadas

```
let _serial = serial;
let _brand = brand;
let _model = model;
let _price = price;
let _taxPercentage = taxPercentage;
```

- Estas variables privadas (`_serial`, `_brand`, `_model`, `_price`, `_taxPercentage`) almacenan los valores iniciales de los parámetros del constructor. Se utilizan para encapsular los datos y protegerlos de cambios externos directos.

5. Uso de `Object.defineProperty` para definir getters y setters

Para cada propiedad, se usa `Object.defineProperty` para definir cómo acceder (get) y cómo modificar (set) los valores:

serial

```
Object.defineProperty(this, 'serial', {
  get: function () {
    return _serial;
  },
  set: function (value) {
    if (!value) throw new EmptyValueException("serial");
    _serial = value;
  }
});
```

- Getter: devuelve el valor de `_serial`.
- Setter: antes de asignar un nuevo valor a `_serial`, valida que no esté vacío.

brand, model y price

- Tienen configuraciones similares a `serial`. Los getters simplemente devuelven el valor almacenado, mientras que los setters validan el valor antes de asignarlo.

taxPercentage

- También tiene un getter que devuelve `_taxPercentage`, y su setter valida que el valor no sea negativo ni undefined.

6. Métodos adicionales en el prototipo de Product

description

```
Object.defineProperty(Product.prototype, 'description', {
  enumerable: true,
  writable: true,
  configurable: false
});
```

Este código añade o modifica la propiedad description en el prototipo de Product con las siguientes características:

- **Enumerable:** La propiedad description aparecerá en las listas de propiedades (como for...in).
- **Writable:** La propiedad se puede modificar.
- **Configurable:** Una vez definida, no se puede eliminar ni cambiar sus atributos (writable, enumerable).

priceWithoutTaxes

```
Object.defineProperty(Product.prototype, 'priceWithoutTaxes', {  
  get: function () {  
    return this.price - (this.price * this.taxPercentage / 100);  
  }  
});
```

- Esta propiedad calcula el precio del producto sin impuestos, restando el valor del impuesto aplicado al precio total.

tax

```
Object.defineProperty(Product.prototype, 'tax', {  
  get: function () {  
    return this.price * this.taxPercentage / 100;  
  }  
});
```

- Calcula el valor del impuesto (taxPercentage) en función del precio actual del producto.


7. Método toString

```
Product.prototype.toString = function () {  
  return "Serial: " + this.serial + " Brand: " + this.brand + " Model: " +  
    this.model + " Price: " + this.price + "€ Tax: " + this.taxPercentage + "%";  
}
```

- Este método sobrescribe el comportamiento por defecto de toString() y devuelve una cadena con una descripción detallada del producto, incluyendo su serial, brand, model, price, y el porcentaje de impuesto (taxPercentage).

8. Propiedad estática IVA

```
Object.defineProperty(Product, 'IVA', {  
  value: 21,  
  writable: false,  
  enumerable: true,  
  configurable: false  
});
```



- Esta propiedad estática (IVA) define el porcentaje de impuesto por defecto para todos los productos como 21%. No es escribible ni configurable, lo que significa que no puede cambiarse después de ser definida.

Resumen:

Este código define una clase Product que:

1. Valida los parámetros de entrada, asegurando que no haya valores vacíos ni inválidos.
2. Encapsula las propiedades serial, brand, model, price, y taxPercentage utilizando getters y setters.
3. Calcula automáticamente el precio sin impuestos y el valor de los impuestos.
4. Proporciona una representación en forma de cadena del producto (toString).
5. Usa una propiedad estática IVA para gestionar el impuesto por defecto (21%).

De esta forma, este constructor ofrece una manera robusta y bien estructurada de gestionar productos con validaciones y cálculos relacionados con impuestos.