
Ice Factory

Clases?

- Agrupar funciones relacionadas en un solo objeto.
- Comercio electrónico, podríamos tener un `cart`objeto que expone una `addProduct`función y una `removeProduct`función.
- Invocar estas funciones con `cart.addProduct()`y `cart.removeProduct()`.

clases en JavaScript se comportan de manera diferente de lo que se espera.

```
// ShoppingCart.js

exportar clase predeterminada ShoppingCart {
  constructor ({db}) {
    this.db = db
  }

  addProduct (producto) {
    this.db.push (producto)
  }

  empty () {
    this.db = []
  }

obtener productos () {
    return Object
      .freeze ([... this.db])
  }

removeProduct (id) {
    // eliminar un producto
  }

// otros metodos

}

// someOtherModule.js

const db = []
const cart = new ShoppingCart ({db})
cart.addProduct ({
  name: 'foo',
  precio: 9.99
})
```

"`new` y `this` [en JavaScript] son una especie de trampa poco intuitiva y extraña".

```
const db = []
const cart = new ShoppingCart ({db})

cart.addProduct = () => 'no!'
// ¡ No hay error en la línea de arriba!

cart.addProduct ({
  name: 'foo',
  precio: 9.99
}) // salida: "¡no!" ¿FTW?
```

- `new` - mutables.
- Puedes **reasignar** un método

- new - heredan prototype el class que se usó para crearlos.
- Los cambios en una clase ' prototype afectan a **todos los** objetos

```
const cart = new ShoppingCart ({db: []})
const other = new ShoppingCart ({db: []})


Carrito de compras. prototipo
   .addProduct = () => 'no!'
// ¡ No hay error en la línea de arriba!


cart.addProduct ({
  name: 'foo',
  precio: 9.99
}) // salida: "¡no!"


other.addProduct ({
  name: 'bar',
  price: 8.88
}) // salida: "¡no!"
```

```
empty () {
    this.db = []
  }
```

- this en JavaScript está vinculado dinámicamente.
- podemos perder la referencia a this

- this ahora se referirá a la button en vez de la cart

```
<button id = "empty">
  Carro vacío
</ botón>


---


document
  .querySelector ('# empty')
  .addEventListener (
    'click',
    cart.empty
  )
```

no hay ningún error en la consola y tu sentido común te dirá que debería funcionar, pero no es así.

Ice Factory?

- una función que crea y devuelve un objeto congelado
- Object.freeze() *congela* un objeto (impide que se le agreguen nuevas propiedades; impide que se puedan eliminar las propiedades ya existentes o puedan ser modificadas; etc)
- El método devuelve el objeto recibido.

```
función predeterminada de exportación makeShoppingCart ({
  db
}) {
  return Object.freeze ({
    addProduct,
    empty,
    getProducts,
    removeProduct,
    // others
  })


function addProduct (producto) {
    db.push (producto)
  }

  function empty () {
    db = []
  }


function getProducts () {
    return Object
      .freeze ([... db])
  }


function removeProduct (id) {
    // eliminar un producto
  }


// otras funciones
}
```

Antes

```
// someOtherModule.js


const db = []
const cart = new ShoppingCart ({db})
cart.addProduct ({
  name: 'foo',
  precio: 9.99
})
```

```
addProduct (producto) {
  this.db.push (producto)
}

empty () {
  this.db = []
}
```

Ahora

```
// someOtherModule.js


const db = []
const cart = makeShoppingCart ({db})
cart.addProduct ({
  name: 'foo',
  precio: 9.99
})
```

```
function addProduct (producto) {
    db.push (producto)
  }

  function empty () {
    db = []
  }
```

Privacidad

```
función makeThing (especificación) {
  const secret = 'shhh!'

return Object.freeze ({
    doStuff
  })

function doStuff () {
    // Podemos usar tanto spec
    // como el secreto aquí
  }
}

// secreto no es accesible aquí

const thing = makeThing ()
thing.secret // undefined
```

Herencia

```
function makeProductList ({productDb}) {
  return Object.freeze ({
    addProduct,
    empty,
    getProducts,
    removeProduct,
    // otros
  )}

  // definiciones para
  // addProduct, etc ...
}
```

```
function makeShoppingCart (productList) {
  return Object.freeze ({
    items:
    productList, someCartSpecificMethod,
    // ...
)}

function someCartSpecificMethod () {
  // code
  }
}
```

Desventajas?

Hacer objetos es más lento y ocupa más memoria que usar una clase.

Bibliografía

https://programmerclick.com/article/51241099738/

Ejercicio :D

```
class Rectangulo {
  constructor (alto, ancho) {
    this.alto = alto;
    this.ancho = ancho;
  }
  // Getter
  get area() {
     return this.calcArea();
   }
  // Método
  calcArea () {
    return this.alto * this.ancho;
  }
}

const cuadrado = new Rectangulo(10, 10);
const cuadrado1 = new Rectangulo(1090, 10);


console.log(cuadrado.area); // 100
```