Practica

Definición

Vamos a manejarnos con un carrito de la compra.

Tenemos la siguiente entrada de datos:

- Un array de productos que hemos comprado.
- Cada producto tiene:
  - El nombre del producto.
  - La cantidad de productos que hemos comprado.
  - El precio.
  - El tipo de IVA que aplica.

Ejemplo:

```
cesta = [
  {
    nombre: "Cerveza",
    cantidad: 1,
    precio: 2,
    tipoIVA: "general"
  },
  {
    nombre: "Galletas",
    cantidad: 3,
    precio: 1,
    tipoIVA: "reducido"
  },
  {
    nombre: "pan",
    cantidad: 4,
    precio: 0.5,
    tipoIVA: "superreducido"
  }
];
```

codepen

Queremos calcular:

- El subtotal de la compra (suma de producto por cantidad sin contar con el IVA).
- El total de la compra (subtotal + IVA).

Calculando el subtotal

Vamos a arrancar por definer nuestros carrito de la compra, y un esqueleto de lo que queremos hacer:

```
class CarritoDeLaCompra {
  constructor() {
    this._cesta = [];
    this._subtotal = 0;
  }

  calculaSubtotal() {}

  set cesta(cesta) {
    this._cesta = cesta;
  }
}
```

Codepen

Lo ideal sería calcular el Subtotal cada vez que cambiamos los elementos de la cesta de la compra:

```
class carritoDeLaCompra {
  constructor() {
    this._cesta = [];
    this._subtotal = 0;
  }

  calculaSubtotal() {
  }

  set cesta(cesta) {
    this._cesta = cesta;
+   this.calculaSubtotal();
  }
}
```

Codepen

Y vamos a exponer una propiedad para mostrar el subtotal:

```
class carritoDeLaCompra {
  constructor() {
    this._cesta = [];
    this._subtotal = 0;
  }

  calculaSubtotal() {
  }

+ get subtotal() {
+   return this._subtotal;
+ }

  set cesta(cesta) {
    this._cesta = cesta;
    this.calculaSubtotal();
  }
}
```

Codepen

Vamos a coger el toro por los cuernos, vamos a aprovechar y repasar lo aprendido en otros módulos: para calcular el subtotal vamos a arrancar implementándolo con un bucle, pasarlo a un reduce para terminar aplicando destructuring.

En la primera versión vamos a iterar e ir sumando en subtotal:

```
class CarritoDeLaCompra {
  constructor() {
    this._cesta = [];
    this._subtotal = 0;
  }

  calculaSubtotal() {
+    this._subtotal = 0;
+    for(let i = 0; i < this._cesta.length; i++) {
+      this._subtotal = this._subtotal + (this._cesta[i].precio * this._cesta[i].cantidad);
+    }
  }
```

Codepen

Y vamos a añadir un código para poder probar lo que hemos hecho:

```
const carrito = new CarritoDeLaCompra();
carrito.cesta = cesta;
console.log("subtotal", carrito.subtotal);
```

codepen

Si ejecutamos este código funciona... ahora estamos en el momento "cárnica" si funciona no lo toques. Si somos buenos desarrolladores podemos ver que ese código se puede hacer un poco complicado de mantener, vamos a ir mejorándolo paso a paso.

En módulos anteriores aprendimos a evitar bucles 'for' en este caso vamos a usar un *reduce*

```
class carritoDeLaCompra {
  constructor() {
    this._cesta = [];
    this._subtotal = 0;
  }

  calculaSubtotal() {
-    this._subtotal = 0;
-    for(i = 0; i < this._cesta.length; i++) {
-      subtotal = subtotal + (this._cesta[i].precio * this._cesta[i].cantidad);
-    }

+    this._subtotal = this._cesta.reduce((acumulado, lineaTicket) =>
+       acumulado + (lineaTicket.cantidad * lineaTicket.precio)
+    , 0);
  }
```

Codepen

Esto está genial, pero, repetimos *lineaTicket*: en tres sitios, ¿No podríamos simplificar esto? Sí, usando destructuring.

```
  calculaSubtotal() {

-    cesta.reduce((acumulado, lineaTicket) =>
+    cesta.reduce((acumulado, { cantidad, precio }) =>
-       acumulado + (lineaTicket.cantidad * lineaTicket.precio)
+       acumulado + (cantidad * precio)
    ,0)
  }
```

Codepen

> Podríamos seguir mejorando este código, por ejemplo sacar el cálculo de una línea de ticket a un método separado, como ejercicio: prueba a sacarlo y comprueba si el código es más fácil de leer.

El código final que se nos queda:

```
cesta = [
  {
    nombre: "Cerveza",
    cantidad: 1,
    precio: 2,
    tipoIVA: "general"
  },
  {
    nombre: "Galletas",
    cantidad: 3,
    precio: 1,
    tipoIVA: "reducido"
  },
  {
    nombre: "pan",
    cantidad: 4,
    precio: 0.5,
    tipoIVA: "superreducido"
  }
```

```
];

class CarritoDeLaCompra {
  constructor() {
    this._cesta = [];
    this._subtotal = 0;
  }

  calculaSubtotal() {
    this._subtotal = this._cesta.reduce(
      (acumulado, { cantidad, precio }) => acumulado + cantidad * precio,
      0
    );
  }

  get subtotal() {
    return this._subtotal;
  }

  set cesta(cesta) {
    this._cesta = cesta;
    this.calculaSubtotal();
  }
}

const carrito = new CarritoDeLaCompra();
carrito.cesta = cesta;
console.log("subtotal", carrito.subtotal);
```

Calculando el total

Para el cálculo del total tenemos que, por un lado, calcular el tipo de IVA a aplicar a cada elemento de la linea de ticket, y después multiplicar dicho factor por precio * cantidad.

Qué podríamos hacer:

Arrancamos por definir las propiedades, getters y métodos que necesitamos.

```
class CarritoDeLaCompra {
  constructor() {
    this._cesta = [];
    this._subtotal = 0;
+    this._total = 0;
  }

  calculaSubtotal() {
    this._subtotal = this._cesta.reduce((acumulado, {cantidad, precio}) =>
       acumulado + (cantidad * precio)
    ,0)
  }

+ calculaTotal() {
+
+ }

+ get total() {
+    return this._total;
+ }

  get subtotal() {
   return this._subtotal;
  }

  set cesta(cesta) {
    this._cesta = cesta;
    this.calculaSubtotal();
+   this.calculaTotal();
  }
}
```

Codepen

Vamos ahora a implementar el cálculo del total:

```
+  calculaFactorIVA(tipoIVA) {
+    switch(tipoIVA) {
+      case 'general':
+         return 1.21;
+      case 'reducido':
+        return 1.1;
+      case 'superreducido':
+        return 1.4;
+    }
+
+    return 1;
+  }

  calculaTotal() {
+     this._total = this._cesta.reduce((acumulado, { cantidad, precio, tipoIVA }) =>
+       acumulado + (cantidad * precio * this.calculaFactorIVA(tipoIVA))
+    , 0)
  }
```

Y para comprobar si esto funciona:

```
const carrito = new CarritoDeLaCompra();
carrito.cesta = cesta;
console.log('subtotal', carrito.subtotal);
+ console.log('total', carrito.total);
```

Codepen

Si tienes ganas de hacer refactoring (mejorar este código):

- Extrae (cantidad * precio) a un método y llámalo **calculaSubTotalLineaTicket(item)**
- Te puedes crear otro método que se llame **calculaTotalLineaTicket(item)** que haga uso del método anterior.

Otro posible ejercicio: podías haber realizado el cálculo del total y el subtotal en una sóla iteración ¿Cómo lo habrías hecho? Evalúa pros y contras de seguir esta aproximación.

Compras en el extranjero - herencia

¿Qué pasa si hacemos el pedido fuera de la Union Europea?... no hay IVA, ¿Qué podríamos hacer?

Podemos extender de la clase *CarritoDeLaCompra*, crearemos una nueva clase llamada *CarritoDeLaCompraExtranjero*.

```
class CarritoDeLaCompraExtranjero extends CarritoDeLaCompra {
  calculaTotal() {
    this._total = this._cesta.reduce(
      (acumulado, { cantidad, precio }) => acumulado + cantidad * precio,
      0
    );
  }
}
```

Y podemos probarlo:

```
+ console.log('** Carrito de la compra normal ***');
const carrito = new CarritoDeLaCompra();
carrito.cesta = cesta;
console.log('subtotal', carrito.subtotal);
console.log('total', carrito.total);

+ console.log('** Carrito de la compra extranjero ***');
+ const carritoExtranjero = new CarritoDeLaCompraExtranjero();
+ carritoExtranjero.cesta = cesta;
```

```
+ console.log('subtotal', carritoExtranjero.subtotal);
+ console.log('total', carritoExtranjero.total);
```

Codepen