Universidad Tecnológica Metropolitana

Estructuras de Datos Aplicadas

ISC. Ruth Betsaida Martínez Domínguez, MGTI

Práctica 1-2

Soberanis Acosta Jimena Monserrat

Desarrollo de Software Multiplataforma

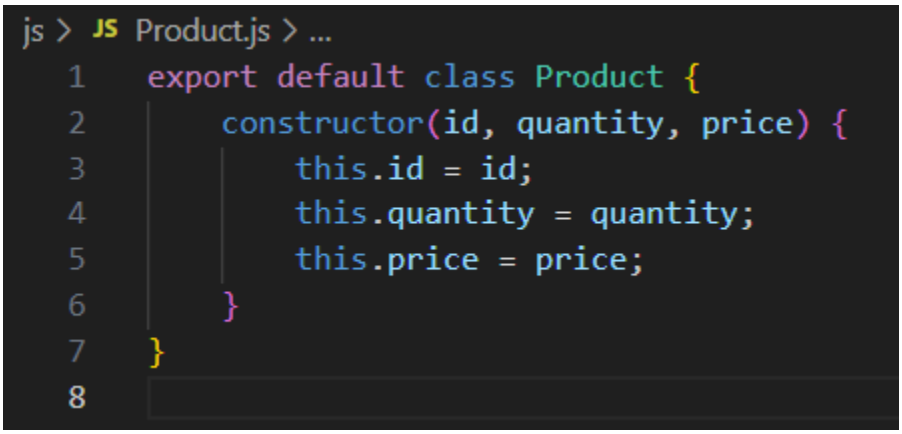
Cuarto Cuatrimestre

4°B

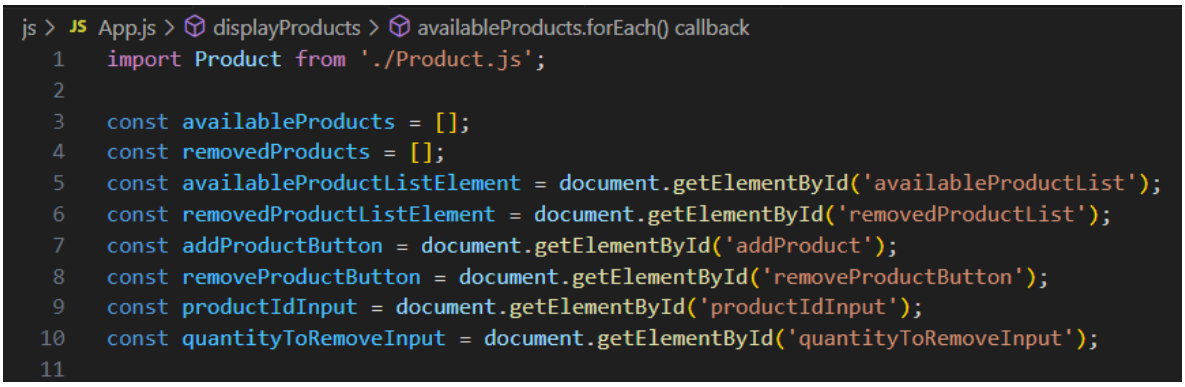
Parcial II

Jueves, 12 de octubre de 2023

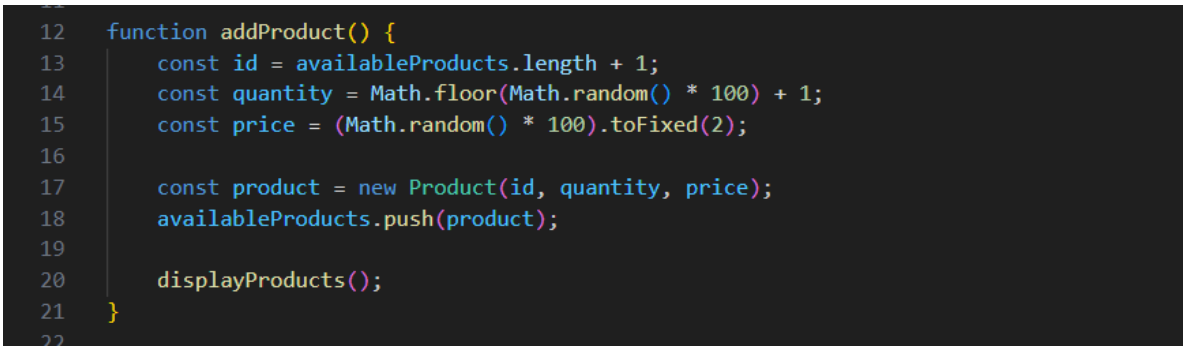
Práctica 1



Comencé con hacer la primera clase llamada Product el cual tiene un constructor con el id, la cantidad y el precio del mismo, usé 'export' para poder usarlo en otros archivos



Ahora, en otro archivo llamado App.js usa 'import' para llamar a la clase Product del archivo Product.js. Después, para comenzar con este parte, se van creando las dos nuevas listas de productos disponibles y retirados, así como se llama a ciertas partes del HTML para agregarles funcionalidades.



Proseguí con el primer método para añadir producto, toma el id, la cantidad y el precio de la clase producto, el id aumenta 1 cada vez, y la cantidad y precio son dos constantes que se crean de manera aleatoria con el método Math.Random. Cada producto que se crea se almacena a la lista de productos disponibles y se usa el método para mostrar productos al final.

```
function displayProducts() {
    availableProductListElement.innerHTML = '';
    removedProductListElement.innerHTML = '';

    availableProducts.forEach((product) => {
        const productItem = document.createElement('li');
        productItem.textContent = `Producto ${product.id} - Cantidad: ${product.quantity}, Precio: $${product.price}`;
        availableProductListElement.appendChild(productItem);
    });

    removedProducts.forEach((product) => {
        const productItem = document.createElement('li');
        productItem.textContent = `Producto ${product.id} - Cantidad: ${product.quantity}, Precio: $${product.price}`;
        removedProductListElement.appendChild(productItem);
    });
}
```

Para poder mostrar los productos en un formato específico, ya sea que se vayan agregando o retirando, toma los elementos de las listas de productos disponibles y retirados, y los pone inicialmente como vacíos. Después por cada una de las listas se crea un foreach que recorre las mismas y crea cada uno de los elementos de la lista con un formato en específico y colocan en la misma.

```
function removeProduct() {
    const productId = productIdInput.value;
    const quantityToRemove = parseInt(quantityToRemoveInput.value);

    const product = availableProducts.find((p) => p.id === parseInt(productId));

    if (product) {
        if (quantityToRemove > 0 && quantityToRemove <= product.quantity) {
            product.quantity -= quantityToRemove;
            if (product.quantity === 0) {
                availableProducts.splice(availableProducts.indexOf(product), 1);
                removedProducts.push(product);
            }
            displayProducts();
            displayMessage(`Producto ${product.id} - ${quantityToRemove} unidades eliminadas`);
        } else {
            displayMessage('Cantidad no válida');
        }
    } else {
        displayMessage('Producto no encontrado');
    }
}
```

Después está la función de remover productos, en donde se obtiene primero el id del producto que se busca y la cantidad que se quiere retirar. Se busca el id entre todos los productos disponibles, si la cantidad a remover es mayor a cero y menor o igual que la cantidad total que hay en la lista, entonces se remueve esa cantidad ingresada de la cantidad original. Entonces, si la cantidad del producto llega a ser 0, ese producto es removido de la lista de productos disponibles y se actualizan ambas listas, en donde se muestra un mensaje de la acción que ocurrió en un formato específico, en casos contrarios a los explicados, salen sus respectivos mensajes de errores.

```
63  function displayMessage(message) {
64      const messageArea = document.getElementById('messageArea');
65      messageArea.textContent = message;
66  }
67
68  addProductButton.addEventListener('click', addProduct);
69  removeProductButton.addEventListener('click', removeProduct);
70
```

Esto mensajes funcionan con la función mostrar mensajes, en donde busca en el HTML esa parte de mensaje, y lo llena con el mensaje en texto.

Por último, están los llamados a los botones de agregar y retirar producto, que ejecutan las respectivas funciones.

index.html > html > body > div#productList > p#messageArea

```
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4       <meta charset="UTF-8">
5       <meta name="viewport" content="width=device-width, initial-scale=1.0">
6       <link rel="stylesheet" href="css/style.css">
7       <title>Almacén de Productos</title>
8   </head>
9   <body>
10      <h1>Almacén de Productos</h1>
11      <div id="agregarProducto">
12          <button id="addProduct">Agregar Producto</button>
13      </div>
14      <div id="retirarProducto">
15          <form>
16              <label for="productIdInput">ID del producto:</label>
17              <input type="number" id="productIdInput" />
18              <br>
19              <label for="quantityToRemoveInput">Cantidad a eliminar:</label>
20              <input type="number" id="quantityToRemoveInput" />
21              <br>
22              <button type="button" id="removeProductButton">Eliminar Producto</button>
23          </form>
24      </div>
25      <div id="productList">
26          <h2>Productos Disponibles</h2>
27          <ul id="availableProductList"></ul>
28          <h2>Productos Retirados</h2>
29          <ul id="removedProductList"></ul>
30          <p id="messageArea"></p>
31      </div>
32      <script src="js/app.js" type="module"></script>
33  </body>
34  </html>
35
```

Previamente a los archivos js hice el HTML respectivo, que al final llama al archivo principal que hace todos los llamados (app.js).

El estilo en CSS se adjunta en el repositorio del GitHub.

Para la aplicación usé Electron:


Almacén de Productos
File Edit View Window Help
Almacén de Productos
Agregar Producto
ID del producto: 1
Cantidad a eliminar: 30
Eliminar Producto
Productos Disponibles
Producto 2 - Cantidad: 56, Precio: $8.03
Producto 3 - Cantidad: 80, Precio: $89.63
Producto 4 - Cantidad: 47, Precio: $53.55
Productos Retirados
Producto 1 - Cantidad: 0, Precio: $80.74
Producto 1 - 30 unidades eliminadas


Almacén de Productos
File Edit View Window Help
Almacén de Productos
Agregar Producto
ID del producto: 2
Cantidad a eliminar: 56
Eliminar Producto
Productos Disponibles
Producto 3 - Cantidad: 80, Precio: $89.63
Producto 4 - Cantidad: 47, Precio: $53.55
Productos Retirados
Producto 1 - Cantidad: 0, Precio: $80.74
Producto 2 - Cantidad: 0, Precio: $8.03
Producto 2 - 56 unidades eliminadas

Práctica 2

Creé el HTML al que iré llamando ciertas partes en la lógica de la práctica

index.html > ...

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Números Aleatorios</title>
    <link rel="stylesheet" href="css/style.css">
</head>
<body>
    <div class="container">
        <h1>Números Aleatorios (10)</h1>
        <button id="generarNumeros">Generar Números Aleatorios</button>
        <div class="fullList">
            <h2>Todos los numeros</h2>
            <ul id="todosLosNumeros"></ul>
        </div>
        <div class="lists">
            <div class="list">
                <h2>Números Pares</h2>
                <ul id="numerosPares"></ul>
            </div>
            <div class="list">
                <h2>Números Impares</h2>
                <ul id="numerosImpares"></ul>
            </div>
        </div>
    </div>
    <script type="module" src="js/app.js"></script>
</body>
</html>
```

js > JS GeneradorNumeros.js > ...

```javascript
export default class GeneradorNumeros{
    static generadorNumeros(contador){
        const numeros = [];
        for (let i =0; i < contador; i++){
            numeros.push(Math.floor(Math.random() * 100) +1);
        }
        return numeros;
    }
}

```

Para la parte lógica comencé con la clase default llamada GeneradorNumeros con un método llamadado generadorNumeros que lo que hace es generar una lista de números aleatorios.

```
js > JS ImparPar.js > ImparPar > filtroImparPar
1  export default class ImparPar{
2      static filtroImparPar(numeros){
3          const numerosPares = numeros.filter(numero => numero % 2 === 0);
4          const numerosImpares = numeros.filter(numero => numero % 2 !== 0);
5
6          return {numerosPares, numerosImpares};
7      }
8  }
```

Después está la clase ImparPar que tiene un método que lo que hace es filtar cada uno de los elementos de la lista de números, si los números al ser divididos en 2 dan un residuo de 0, entonces se guarda como numeroPares, y si da otro residuo diferente a 0 (decimales) se guarda en numerosImpares.

```
js > JS MostrarNumeros.js > MostrarNumeros > displayNumbers
1  export default class MostrarNumeros {
2      static displayNumbers(numeros, elementId) {
3          const list = document.getElementById(elementId);
4          list.innerHTML = '';
5
6          const comas = numeros.join(", ");
7
8          const listItem = document.createElement('li');
9          listItem.textContent = comas;
10         list.appendChild(listItem);
11     }
12 }
```

Después está la clase MostrarNumeros, el cual tiene un método que toma la lista de números y el elementoId, este elementoId se convierte en una constante llamada list, y este está vacío al inicio. Luego, está la variable comas que lo que hace es juntar a cada número entre “, ”. Para crear la lista de cada tipo de nuevo, se crea la variable listItem que crea un elemento ‘li’ y se junta los datos de la lista junto a las comas.

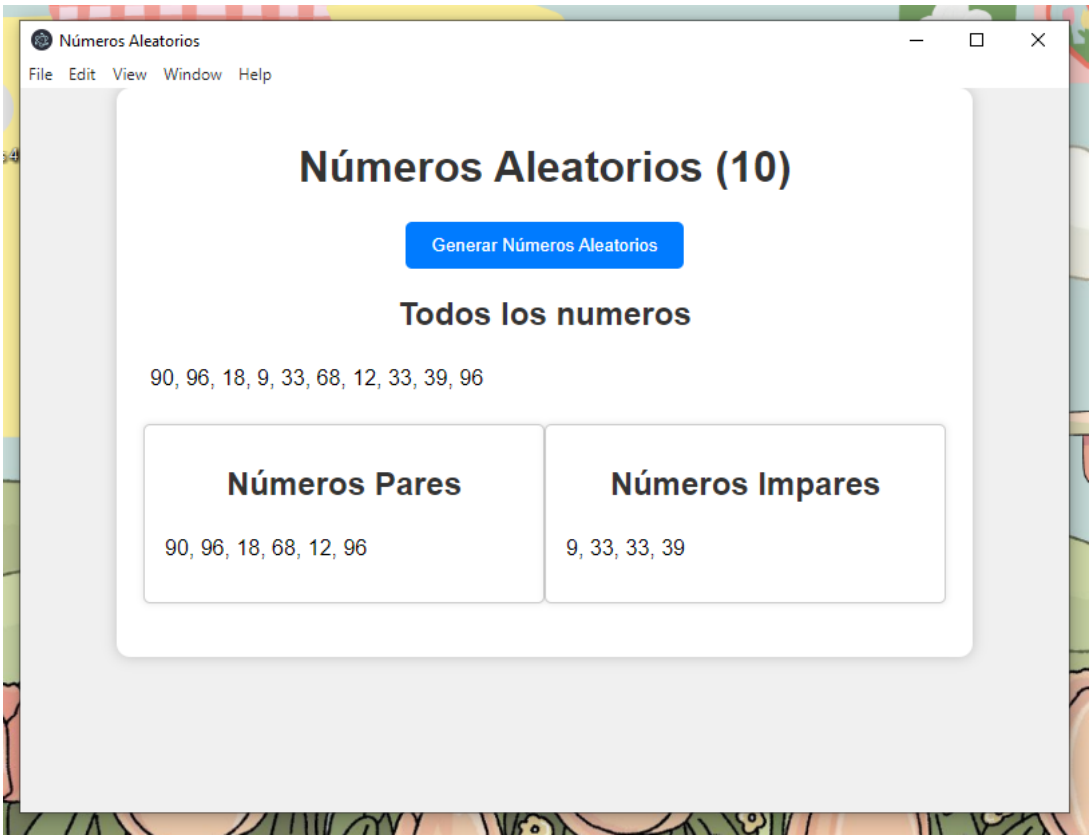
```
js > JS App.js > ...
1  import GeneradorNumeros from './GeneradorNumeros.js';
2  import ImparPar from './ImparPar.js';
3  import MostrarNumeros from './MostrarNumeros.js';
4
5  document.getElementById("generarNumeros").addEventListener("click", () => {
6      const generarNumeros = GeneradorNumeros.generadorNumeros(10);
7
8      MostrarNumeros.displayNumbers(generarNumeros, "todosLosNumeros");
9      const { numerosPares, numerosImpares } = ImparPar.filtroImparPar(generarNumeros);
10
11     MostrarNumeros.displayNumbers(numerosPares, "numerosPares");
12     MostrarNumeros.displayNumbers(numerosImpares, "numerosImpares");
13 });
```

Por último, está el archivo Apps.js, el principal que usa el HTML, este archivo importa a los demás para poder usarlos. Entonces, se llama al botón con el id generarNumeros que lo que hace es crear al inicio una variable que genera 10 numeros aleatorios, este número puede variar.

Después se llama a la clase MostrarNumeros que utiliza el método de mostrarNumeros, el cual muestra una lista con todos los números generados, luego, se filtran estos entre par e impar. Al final se muestran cada lista filtrada.

El estilo en CSS se adjunta en el repositorio del GitHub.

Para la aplicación usé Electron:



Para cada práctica (1-2), el orden de los archivos y carpetas quedaron así: