Práctica de JavaScript

Operaciones con productos y carrito de compras

Objetivo: Crear un conjunto de funciones en JavaScript que permitan simular operaciones de altas, bajas, cambios y consultas en una app de e-commerce.

Descripción de la aplicación

Se desea tener un conjunto de funciones en JavaScript que permitan simular operaciones de altas, bajas, cambios y consultas para la parte de productos y el carrito de compras de una app de e-commerce.

Esta práctica contiene los siguientes puntos:

1. Consideraciones
2. Configuración inicial
3. Funciones de Productos
4. Funciones del Carrito de Compras
5. Funciones para Manejo de Datos
6. Búsqueda de usuarios por diferentes criterios
7. Pruebas y ejemplos
8. Evaluación

Consideraciones

Considerar los siguientes aspectos para la implementación de la práctica:

1. Utilizaremos el archivo **home.html** de la **práctica 1**.
2. Mostraremos la funcionalidad de nuestro conjunto de funciones de **manera visual** y desde **la consola del navegador**.
3. En esta práctica no se espera aun la interacción con la página web (solo mostrar información), no se habilitarán botones, ni modales, etc…

Configuración inicial

Crea los siguientes archivos de **JavaScript**:

- products.js
- shopping_cart.js
- utils.js
- data_handler.js
- index.js

En tu archivo **home.html** de la **práctica 1**, carga los scripts de JavaScript en cualquier lugar (en head o en body).

Funciones de Productos

Para poder utilizar correctamente los productos en nuestra aplicación, necesitamos una manera de representarlos en objetos de **JavaScript**. Para esto en nuestro archivo de **products.js**, agregaremos 2 clases: **Product** y **ProductException**.

La clase **Product** debe ser capaz de guardar los siguientes datos:

```
[{
    "uuid":"df2008a5-1c40-4dd1-9db7-8aacc03ae2fb",
    "title": "Platano",
    "description":"Los mejores platanos de México, directo desde Tabasco.",
    "imageUrl":"https://images.freeimages.com/images/large-previews/4ec/banana-s-
1326714.jpg",
    "unit": "pieza",
    "stock": 15,
    "pricePerUnit": 3.6,
    "category":"Fruta"
}]
```

Necesitaremos **getters** y **setters** que hagan las validaciones adecuadas (tirando excepciones de tipo **ProductException**), como no permitir Strings vacios, números negativos, etc...

Para el UUID, lanzaremos una excepción en el **setter** para solo permitir el uso interno, además, utilizaremos el siguiente código para auto-generar los **UUIDs** al crear nuestros productos:

```
function generateUUID() {
    return 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, c => {
        let r = Math.random() * 16 | 0;
        let v = c == 'x' ? r : (r & 0x3 | 0x8);
        return v.toString(16);
    });
}
```

Copia este código en el archivo **utils.js**.

Para poder interactuar con el servidor, debemos de ser capaz de interpretar objetos externos como productos, para lo cual crearemos las siguientes funciones estáticas:

- **createFromJson**(jsonValue): Esta función debe convertir el **String** de **JSON** recibido en una nueva instancia de producto (utilizando la clase **Product**).
- **createFromObject**(obj): Esta función debe convertir el **objeto** recibido en una nueva instancia de producto (utilizando la clase **Product**) y debe ser capaz de ignorar todos aquellos valores que no pertenezcan a la clase **Product**.
- **cleanObject**(obj): Esta función debe limpiar el **objeto** recibido de todos aquellos valores que no pertenezcan a la clase **Product**.

Funciones del Carrito de Compras

Esta tal vez sea **la parte más difícil de la práctica**, la dejamos al inicio para que consultes al profesor en caso de que lo requieras.

En nuestro archivo de **shopping_cart.js**, agregaremos 3 clases: **ShoppingCart**, **ProductProxy** y **ShoppingCartException**.

Para referenciar a los productos que agregamos a nuestro carrito de compras, utilizaremos la clase **ProductProxy**, en la cual guardaremos el **UUID** del producto y la cantidad a comprar.

Dentro de nuestra clase de **ShoppingCart**, guardaremos 2 arreglos uno para los **proxies** y otro de **productos**, en donde guardaremos una copia de los verdaderos productos a comprar. Debemos validar que el usuario no pueda modificar directamente el arreglo de **proxies**, en lugar de eso agregaremos los siguientes métodos:

- **addItem**(productUuid, amount): Esta función debe agregar al carrito un nuevo **ProductProxy**, o en caso de que exista ya el producto, actualizar la cantidad a la suma de ambos valores (cantidad original + cantidad nueva).
- **updateItem**(productUuid, newAmount): Esta función debe actualizar el producto correspondiente a la nueva cantidad. Si el nuevo valor es negativo, mandamos error, si es igual a 0, eliminamos el producto del carrito y si es mayor a 0, entonces actualizamos al nuevo valor.
- **removeItem**(productUuid): Esta función debe eliminar el **producto** correspondiente.
- **calculateTotal**(): Esta función debe calcular el valor total de la compra, utilizando la cantidad correspondiente a cada producto y el valor unitario del mismo.

Funciones para Manejo de Datos

Para esta parte crearemos en el archivo de data_handler.js un arreglos de productos, que será nuestra lista oficial de productos a la venta.

Debemos agregar los métodos correspondientes para leer (**getProducts**(), **getProductById**(**uuid**)), para crear (**createProduct**(**product**)), para actualizar (**updateProduct**(**uuid**, **updatedProduct**)) y para eliminar (**deleteProduct**(**uuid**)).

Además del CRUD, de manera opcional puedes agregar un método para búsqueda por categoría o por nombre del producto. Para esto crearemos **findProduct**(**query**), en donde query es un String, en el cual pondremos la categoría y el nombre a buscar en el siguiente formato **"<category>: <title>"**.

En caso de que el usuario pase solo la categoría (**"<category>:"**) haremos la búsqueda de los productos que contengan esa cadena dentro de su categoría. De manera similar si el usuario pasa solo el nombre (**"<title>"**) haremos la búsqueda de los productos que contengan esa cadena dentro de su nombre.

Para el caso combinado, debemos dividir la consulta en las 2 partes correspondientes y aplicar ambos filtros para la lista de productos.

Pruebas y Ejemplos

En **index.js** mandaremos llamar nuestras diferentes funciones para validar que todo se ejecute según lo planeado.

Para las pruebas de la clase de **productos**, poblaremos la lista de productos en **data_handler.js** agregando valores similares a los que teníamos en la **práctica 1**, debemos agregar por lo menos 4 elementos, después actualizar alguno, posteriormente eliminar alguno y por último en caso de haber implementado la parte de búsqueda, hacer 3 búsquedas una solo por categoría, otra solo por nombre y otra combinada.

Para las pruebas de la clase del carrito de compras, agregaremos 3 elementos de nuestra lista de productos a un nuevo objeto **ShoppingCart**, después actualizaremos alguno de ellos y eliminaremos alguno de ellos. También calcularemos el total, para lo cual necesitamos poblar los valores reales de productos en el carrito de compras.

De manera opcional puedes cargar los elementos directamente en el contenedor principal apoyándote de la propiedad **innerHTML**. Esto requiere crear un método que mapee los valores del producto a un **String** con el contenido de HTML correspondiente a una tarjeta como las de la **práctica 1**, pregúntale al profesor por sintaxis de DOM (tema por ser visto posteriormente durante las sesiones).

Evaluación

Sección	Puntuación
Clase Product	15 puntos
Clase ShoppingCart	15 puntos
Uso de excepciones y validaciones	10 puntos
Funciones estáticas de Product	20 puntos
CRUD del carrito de compras	20 puntos
CRUD en data_handler.js	15 puntos
Pruebas y ejemplos	5 puntos
Uso de productListToHtml	5 puntos
Búsquedas por categoría y/o nombre	10 puntos

Máxima calificación de la práctica: 100 puntos.

Entregables:

- Recuerda entregar tanto el archivo HTML de home, así como los 5 archivos correspondientes de JavaScript.
- Debes además adjuntar un reporte con las evidencias y la rúbrica contestada de acuerdo a lo que se alcanzó a entregar.
- En el reporte añadir conclusiones, cosas por mejorar, temas que costaron trabajo, etc…