Ejercicio: Gestor de pedidos online con estructuras avanzadas de datos

Resultado de aprendizaje en alternancia:

R.A 6: Escribe programas que manipulen información seleccionando y utilizando tipos avanzados de datos.

Tareas:

A) Implementar programas que utilicen arrays unidimensionales y multidimensionales, comprendiendo su uso en distintos contextos de desarrollo.
B) Identificar y aplicar librerías estándar del lenguaje de programación (como Collections en Java) para gestionar datos de forma eficiente.
C) Crear y manipular listas dinámicas para almacenar y procesar información de manera flexible y optimizada.
D) Diseñar e implementar clases genéricas que permitan manejar distintos tipos de datos sin perder reutilización y flexibilidad en el código.
E) Desarrollar métodos genéricos que permitan operar con distintos tipos de datos sin necesidad de reescribir código específico para cada tipo.
F) Aplicar buenas prácticas en la manipulación de estructuras de datos avanzadas, garantizando eficiencia y legibilidad en el código.

Enunciado:

Contexto del ejercicio:

Una pequeña empresa de reparto necesita digitalizar la gestión de sus pedidos. Para ello, te han pedido desarrollar una serie de funciones que les permitan procesar información de sus envíos, clientes y productos de forma eficiente y flexible.

Deberás realizar distintas tareas relacionadas con el almacenamiento, manipulación y procesamiento de los datos, usando estructuras avanzadas y aplicando buenas prácticas.

Parte 1 – Gestión de zonas y tarifas (A)

La empresa divide su servicio en distintas **zonas de envío**, cada una con una tarifa y un número de pedidos asignados en los últimos 7 días.

Crea un array bidimensional en el que cada fila represente una zona. Cada zona tendrá:

- Nombre de la zona (string)
- Tarifa (número decimal)
- Array con el número de pedidos diarios durante una semana (array de 7 enteros)

Implementa una función que calcule la **media de pedidos por zona** y determine cuál es la zona más activa.

(A) Implementar programas que utilicen arrays unidimensionales y multidimensionales…

Parte 2 – Uso de librerías para ordenación y búsqueda (B)

Utilizando **la librería `lodash`**, ordena las zonas en función de su tarifa, de menor a mayor, y busca la zona que tenga una tarifa exacta (por ejemplo, 4.5).

(B) Identificar y aplicar librerías estándar del lenguaje…)

Parte 3 – Gestión dinámica de productos (C)

Crea una **lista dinámica de productos** (array que irá creciendo) donde cada producto tendrá:

- ID
- Nombre
- Precio
- Stock disponible

Implementa funciones que permitan:

- Añadir productos
- Eliminar un producto por ID
- Actualizar el stock de un producto
- Filtrar productos cuyo stock esté por debajo de un cierto umbral

(C) Crear y manipular listas dinámicas…)

Parte 4 – Clase genérica para manejo de colecciones (D)

Diseña una clase genérica `CollectionManager<T>` que permita gestionar cualquier tipo de colección de objetos (por ejemplo, productos, clientes, etc.).

Debe incluir métodos como:

- `add(item)`
- `removeById(id)`
- `findById(id)`
- `listAll()`

(D) Diseñar e implementar clases genéricas…)

Parte 5 – Método genérico para comparar objetos (E)

Crea un **método genérico** llamado `compareByKey(arr, key)` que reciba un array de objetos y una clave, y devuelva los elementos ordenados por esa clave.

Este método debe funcionar con cualquier tipo de objeto (productos, clientes, pedidos...).

(E) Desarrollar métodos genéricos que permitan operar con distintos tipos de datos…)

Parte 6 – Mejora y revisión del código (F)

🔻 Refactoriza las funciones anteriores aplicando buenas prácticas:

- Nombres descriptivos
- Separación de lógica en funciones
- Comentarios breves que expliquen los pasos clave
- Uso de estructuras adecuadas para mejorar la legibilidad y eficiencia (por ejemplo, `Map` en lugar de arrays cuando proceda)

(F) Aplicar buenas prácticas en la manipulación de estructuras de datos avanzadas…)

Estructura recomendada del proyecto

```
mi-proyecto/
├── index.html      # Página principal
├── main.js         # Archivo principal con la lógica del programa
├── utils.js        # Funciones de utilidad (opcional)
├── images          # Carpeta para las imágenes
└── README.md       # Instrucciones y explicación de tu solución
```

Bloques sugeridos del código

(A) Arrays unidimensionales y multidimensionales

```
// Crear un array de pedidos
const pedidos = [
  { id: 1, cliente: "Ana", productos: ["camiseta", "gorro"], total: 30 },
  { id: 2, cliente: "Luis", productos: ["zapatillas", "sudadera"], total: 85 }
];

// Crear una tabla multidimensional con categorías y productos
const catalogo = [
  ["ropa", "camiseta", "pantalón"],
  ["calzado", "zapatillas", "botas"]
];
```

(B) Uso de librerías estándar (`lodash`)

```
// Requiere instalación previa: npm install lodash
import _ from "lodash";

// Ordenar pedidos por total
const pedidosOrdenados = _.orderBy(pedidos, ['total'], ['desc']);
```

(C) Lista dinámica

```
// Funciones para añadir, eliminar y filtrar pedidos
function agregarPedido(pedido) {
  pedidos.push(pedido);
```

```
}

function eliminarPedidoPorId(id) {
  // ...
}
```

(D) Clase genérica (ej. almacenamiento de datos)

```
class GestorDatos {
  constructor() {
    this.elementos = [];
  }

  agregar(elemento) {
    this.elementos.push(elemento);
  }

  obtenerTodos() {
    return this.elementos;
  }
}
```

(E) Método genérico

```
function ordenarPorClave(array, clave) {
  return [...array].sort((a, b) => (a[clave] > b[clave] ? 1 : -1));
}
```

(F) Buenas prácticas

- Usa nombres descriptivos: `cliente`, `agregarPedido`, `ordenarPorClave`, etc.
- Separa la lógica en funciones reutilizables.
- Evita repeticiones innecesarias.

README.md (a completar por el alumno)

Describe aquí las decisiones que has tomado:

- ¿Por qué usaste arrays y no otro tipo de estructura?
- ¿Qué ventajas tiene la clase GestorDatos?
- ¿Cómo reutilizaste el método de ordenación?
- ¿Qué problemas encontraste y cómo los solucionaste?

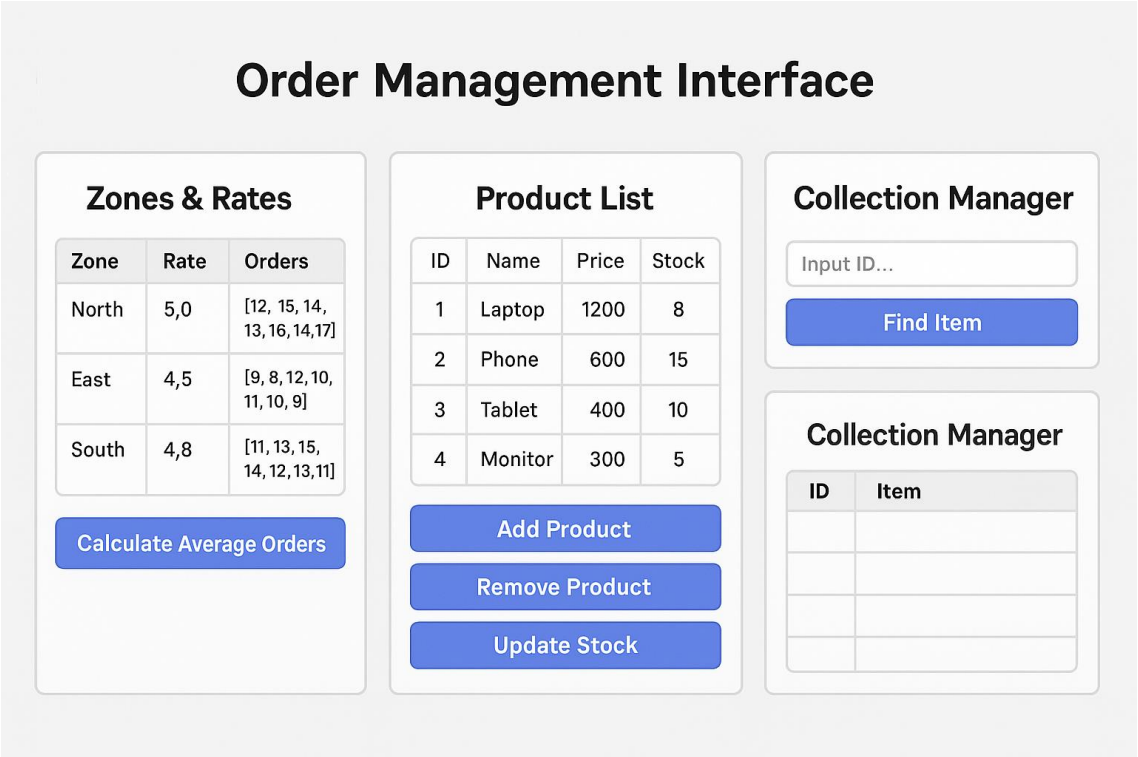
Opcional: mini-test para validar tu código

```
console.log("Pedidos ordenados:", ordenarPorClave(pedidos, "total"));
console.log("Catálogo:", catalogo);
```

Ideas para la interfaz del programa

A continuación, tienes varias ideas para crear las pantallas de tu aplicación. No es necesario que sean como las propuestas, pero pueden servirte como orientación.

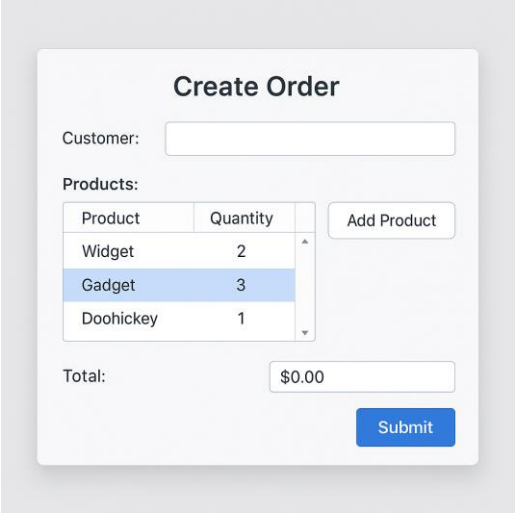
Pantalla Principal:



Incluye:

- Una tabla de pedidos a la izquierda (con ID, cliente, productos, total...).
- Un listado de productos en el centro, con posibilidad de agregar o editar.
- Una sección de gestión de zonas y tarifas a la derecha, simulando los arrays multidimensionales.
- Controles para filtrar, ordenar, añadir elementos y realizar operaciones dinámicas.

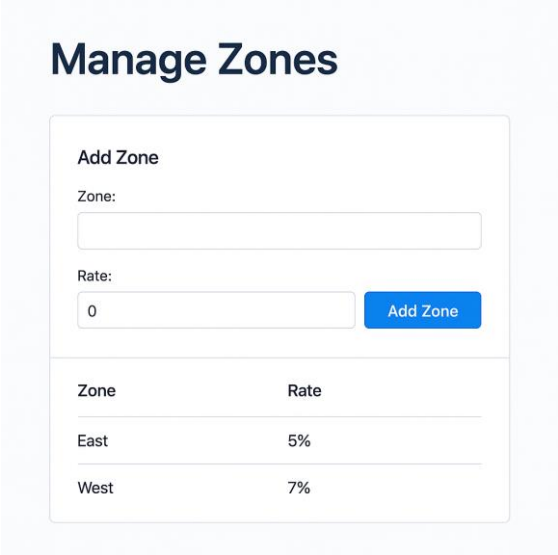
Pantalla de de creación de pedidos:



Incluye:

- **Formulario de cliente**: Campos para introducir nombre, email y dirección.
- **Selección de productos**: Lista desplegable o botones con productos disponibles.
- **Resumen del pedido**: Aparece una tabla dinámica con productos añadidos (como "Gadget"), unidades y subtotales.
- **Total del pedido**: Cálculo automático en la parte inferior.
- **Botón "Submit"**: Azul destacado, enfocado en la acción principal.

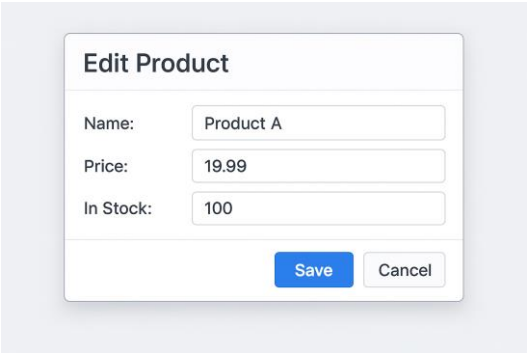
Pantalla de gestión de zonas y tarifas:



Incluye:

- **Formulario para añadir una nueva zona**: Nombre de la zona y tarifa correspondiente.
- **Tabla clara y organizada**: Muestra todas las zonas creadas junto con sus tarifas, permitiendo una visualización inmediata de los datos.
- **Botones de acción**: Como editar o eliminar zonas específicas.

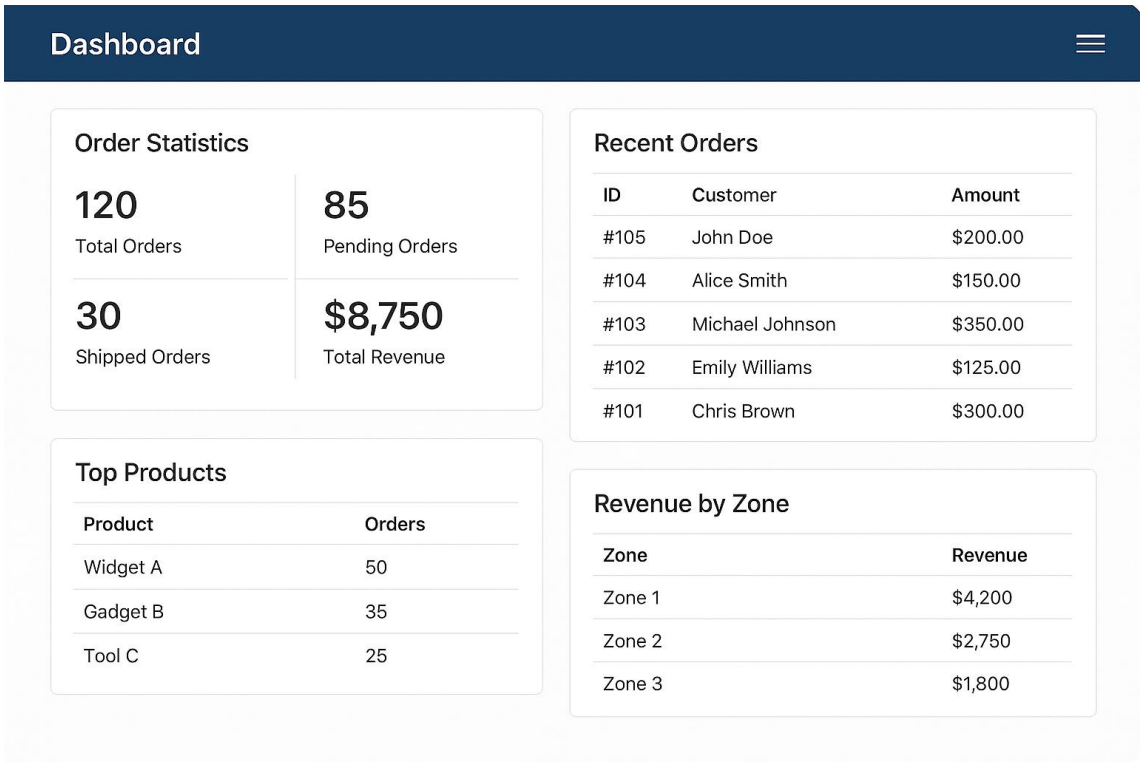
Pantalla de edición de productos:



Incluye:

- **Formulario de edición** con campos bien definidos:
  - Nombre del producto.
  - Precio.
  - Cantidad en stock o disponibilidad.
- **Botones de acción**: "Guardar" (Save) en azul y "Cancelar" (Cancel) en gris para reforzar la toma de decisiones.

Pantalla de panel resumen:



Incluye:

- **Order Statistics**: Grandes contadores visuales con el número total de pedidos, pedidos pendientes, enviados, etc.
- **Recent Orders**: Tabla clara con datos clave (cliente, fecha, estado).
- **Top Products**: Ranking con los productos más solicitados y su cantidad.
- **Revenue by Zone**: Gráfico o tabla que muestra ingresos según zonas de envío (ideal para asociarlo con arrays multidimensionales o estructuras jerárquicas de datos).

Entrega esperada:

- Código completo en un archivo `.js` bien estructurado junto con una página web en formato html y los recursos necesarios en una carpeta, como por ejemplo las imágenes usadas.
- Comentarios explicativos en puntos clave.
- Uso de al menos una librería externa (`lodash`).
- Clases y métodos genéricos correctamente implementados.

Rúbrica de Evaluación

Criterio	Tareas vinculadas	Excelente (4)	Bien (3)	Suficiente (2)	Insuficiente (1)
1. Implementación de arrays uni y multidimensionales	A	Utiliza arrays con estructuras lógicas y bien organizadas, e implementa cálculos correctos y eficientes.	Utiliza arrays adecuados y resuelve los cálculos con algún pequeño error o ineficiencia.	Usa arrays básicos sin aprovechar su potencial; errores en los cálculos.	No utiliza arrays correctamente o no resuelve la parte correspondiente.
2. Uso de librerías estándar para gestión de datos (lodash)	B	Aplica `lodash` correctamente para ordenar y buscar, eligiendo métodos adecuados.	Usa `lodash` pero con algún método innecesario o confuso.	Uso muy limitado o incorrecto de la librería.	No utiliza librerías o el código no funciona.
3. Creación y manipulación de listas dinámicas	C	Implementa correctamente todas las funciones (añadir, eliminar, actualizar, filtrar) con una lista dinámica.	Implementa la mayoría de funciones de forma funcional y clara.	Las funciones tienen errores o faltan partes clave.	No implementa las funciones o no funcionan.
4. Diseño de una clase genérica reutilizable	D	Diseña una clase flexible, bien estructurada, reutilizable para diferentes tipos, aplicando tipado y buenas prácticas.	Diseña la clase genérica correctamente, aunque con margen de mejora en reutilización o estructura.	Clase funcional pero con errores conceptuales o poca generalización.	Clase ausente o totalmente incorrecta.
5. Desarrollo de un método genérico para ordenación por clave	E	Crea un método versátil y funcional que trabaja con cualquier tipo de objeto.	Método funcional, aunque algo limitado a ciertos tipos o sin validar errores.	Método poco reutilizable o con errores lógicos.	No se desarrolla o no funciona.
6. Aplicación de buenas prácticas y legibilidad	F	Código limpio, bien estructurado, nombres claros, lógica modular y buen uso de estructuras.	Código mayormente claro, con algún aspecto mejorable en modularidad o estructura.	Código confuso o con estructuras mal elegidas.	Código desorganizado, con errores de estilo graves.
7. Comentarios y explicación de decisiones técnicas	Todas (en conjunto)	Comentarios claros, útiles y bien distribuidos. Justifica elecciones técnicas (uso de arrays, clases, métodos, etc).	Comentarios presentes y correctos, aunque poco explicativos en algunos casos.	Comentarios escasos o poco útiles.	Sin comentarios o muy vagos.
8. Funcionamiento general y coherencia del programa	Todas (en conjunto)	El programa completo funciona, cada parte está bien	El programa funciona en su mayor parte, con leves	Varias partes no funcionan o están desconectadas.	El programa no se ejecuta correctamente o

		integrada con las demás.	desconexiones entre componentes.		carece de coherencia.

Escala de puntuación:

- **32 – 28 puntos** → *Excelente*
- **27 – 22 puntos** → *Bien*
- **21 – 16 puntos** → *Suficiente*
- **15 o menos** → *Insuficiente*