Parcial Práctico de JavaScript —

Contexto

Vas a construir un **sistema de gestión de entregas** para una empresa de logística que maneja **pedidos en tiempo real**.
El sistema debe ser capaz de:

- Recibir pedidos asincrónicamente.
- Validar y agruparlos.
- Asignar repartidores de manera inteligente.
- Detectar situaciones anómalas (errores, exceso de carga).
- Aplicar algoritmos para priorizar entregas.

Enunciado

1. Clases, prototipos y errores personalizados

- Crear una clase `Pedido` con propiedades: `id`, `cliente`, `producto`, `estado` (`pendiente`, `asignado`, `en ruta`, `entregado`, `cancelado`).
- Agregar un método al **prototipo** (`Pedido.prototype.cancelar`) que cambie el estado a `"cancelado"`.
- Crear una clase de error `EstadoInvalidoError` que se lance si alguien intenta poner un estado no permitido.

---

2. Gestión de pedidos (lógica interna)

Crear la clase `GestorPedidos` que:

- Mantenga los pedidos en un array privado `#pedidos`.

- Métodos obligatorios:
    - `agregarPedido(pedido)`
        - Valida que no haya duplicados (`===`, no `==`).
        - Si se intenta duplicar, lanza un `Error("Pedido duplicado")`.
    - `agruparPorEstado()`
- Devuelve un objeto con la forma:

```
{
  pendiente: [...],
  asignado: [...],
  enRuta: [...],
  entregado: [...],
  cancelado: [...]
}
```

        -
    - `asignarPedidos(repartidores)`
        - Recibe un array de nombres.
        - Implementa un **algoritmo round-robin** que distribuya los pedidos pendientes entre repartidores.
        - Si un repartidor recibe más de 3 pedidos en total, mover los pedidos excedentes a una lista de espera (`listaEspera`).

---

3. Asincronía + Timers

- Simular la llegada de pedidos con `setInterval`: cada 3 segundos se agrega un pedido nuevo con un producto aleatorio.
- Usar `Promise.all` para simular que 3 pedidos “en ruta” se entregan en paralelo después de tiempos distintos (`setTimeout`).
- Si en cualquier momento hay más de **10 pedidos pendientes**, mostrar un warning en consola:
  `"⚠️ Demasiados pedidos sin asignar"`

---

4. Closures y this

- Dentro de `GestorPedidos`, implementar un método `contadorPedidosPorCliente(cliente)` que devuelva un **closure**.
    - Este closure recuerda cuántos pedidos ha hecho el cliente, incluso después de que termine la función.
- Explicar en comentarios por qué este código falla:

```
const gestor = new GestorPedidos();
console.log(gestor.totalPedidos()); // ❌ Error
function totalPedidos() {
  return this.#pedidos.length;
}
```

- Y reescribirlo correctamente dentro de la clase usando `this`.

---

5. Validaciones avanzadas y coerción

- Implementar un método `validarEntradaPedido(obj)` que:
    - Reciba un objeto genérico desde la API.
    - Use **coerción explícita** para transformar campos (`Number`, `String`, `Boolean`).
    - Si falta un campo obligatorio (`id`, `cliente`, `producto`), lanzar un `Error("Datos incompletos")`.
- Demostrar con ejemplos la diferencia entre `==` y `===` al validar IDs de pedidos (`"5" == 5` vs `"5" === 5`).

---

6. Nueva lógica — Sistema de Prioridades

Implementar dentro de `GestorPedidos`:

- Método `priorizarPedidos()` que:

    - Clasifique pedidos en **alta**, **media** y **baja** prioridad.
    - Criterios:
        - Alta: pedidos de clientes VIP (definidos en una lista `clientesVIP`).
        - Media: pedidos con productos que tardan más de 30 min en prepararse (definidos en una lista `productosLentos`).
        - Baja: el resto.
    - Devuelva los pedidos ordenados de forma que los de **alta prioridad siempre se atiendan primero**, seguidos de media y luego baja.
- Extra: si hay más de 2 pedidos de alta prioridad en cola, suspender la asignación de baja prioridad hasta que se liberen.

Criterios de Evaluación

✅ Correcto uso de **clases, prototipos, módulos y errores personalizados**.
✅ Aplicación de **asincronía avanzada (Promise.all, setInterval, setTimeout)**.
✅ Uso de **closures y this** de forma clara.
✅ Lógica de **algoritmos no triviales**: round-robin, lista de espera, prioridades dinámicas.
✅ Validaciones sólidas con **coerción y manejo de errores**.
✅ Código bien comentado y con explicaciones de decisiones.