**Dificultad Media**

**1. Sistema de Construcción de Menús para Restaurantes (Builder)**

**Contexto:** Una cadena de restaurantes necesita un sistema flexible para crear menús personalizados según el tipo de cliente (infantil, vegetariano, ejecutivo). Cada menú debe construirse paso a paso, permitiendo agregar componentes opcionales y validar que los menús tengan los elementos obligatorios.  
**Requisitos:**

* Cada menú debe tener un plato principal, una bebida y opcionalmente postre, aperitivo o ambos.
* Algunos restaurantes pueden incluir componentes especiales (ej: pan artesanal, sopa del día).
* El proceso de construcción debe evitar menús inválidos (ej: sin plato principal).
* Debe ser fácil agregar nuevos tipos de menús en el futuro.  
  **Funcionalidades:**
* Métodos para agregar cada componente (plato principal, bebida, postre, etc.).
* Validación de que el menú esté completo antes de obtener el objeto final.
* Posibilidad de crear menús con configuraciones predefinidas (ej: menú vegetariano estándar).

**2. Logger Centralizado para Aplicación Distribuida (Singleton)**

**Contexto:** Una aplicación distribuida de comercio electrónico necesita un único logger accesible desde todos los módulos (catálogo, carrito, usuarios, pagos). El logger debe ser thread-safe y lazy-loaded.  
**Requisitos:**

* Solo debe existir una instancia del logger en toda la aplicación.
* Debe ser accesible globalmente sin necesidad de pasar referencias.
* Debe soportar escritura en archivo y en consola simultáneamente.
* Debe ser eficiente para evitar cuellos de botella en operaciones concurrentes.  
  **Funcionalidades:**
* Método para registrar eventos con diferentes niveles (INFO, WARN, ERROR).
* Configuración centralizada del nivel de log (ej: solo logs de nivel ERROR en producción).
* Rotación de archivos de log para evitar que crezcan indefinidamente.

**3. Fábrica de Procesadores de Pagos (Factory Method)**

**Contexto:** Un sistema de pagos en línea debe procesar transacciones usando diferentes proveedores (PayPal, Stripe, Tarjeta de Crédito). Cada proveedor tiene su propia lógica de procesamiento.  
**Requisitos:**

* El sistema debe decidir en runtime qué procesador usar basado en el método de pago seleccionado por el usuario.
* Todos los procesadores deben implementar la misma interfaz para que el código cliente sea uniforme.
* Debe ser fácil agregar nuevos procesadores en el futuro.  
  **Funcionalidades:**
* Creación de procesadores específicos según el método de pago.
* Validación de parámetros específicos para cada procesador (ej: CVV para tarjetas).
* Manejo de errores específicos de cada proveedor.

**4. Clonador de Configuraciones de Servidor (Prototype)**

**Contexto:** Un sistema de deployment en la nube necesita clonar configuraciones base para personalizarlas en diferentes entornos (desarrollo, testing, producción).  
**Requisitos:**

* Las configuraciones son objetos complejos con múltiples parámetros (CPU, RAM, almacenamiento).
* Las copias deben ser independientes del original.
* Debe ser eficiente para evitar recrear configuraciones desde cero.  
  **Funcionalidades:**
* Creación de copias exactas de configuraciones base.
* Personalización de parámetros específicos en cada entorno.
* Validación de que las configuraciones clonadas son válidas.

**5. Sistema de Temas para Aplicación de Escritorio (Abstract Factory)**

**Contexto:** Una aplicación de escritorio debe soportar múltiples temas visuales (claro, oscuro, high-contrast). Cada tema debe proporcionar componentes consistentes.  
**Requisitos:**

* Cada tema debe proveer botones, campos de texto, ventanas y otros componentes con estilos coherentes.
* El cambio de tema debe ser en runtime sin reiniciar la aplicación.
* Debe ser fácil agregar nuevos temas en el futuro.  
  **Funcionalidades:**
* Creación de familias de componentes UI para cada tema.
* Intercambio dinámico de temas.
* Previsualización de temas antes de aplicarlos.

**Dificultad Alta**

**6. Constructor de Queries para ORM Avanzado (Builder)**

**Contexto:** Un ORM (Object-Relational Mapping) avanzado necesita construir queries SQL complejos con múltiples joins, condiciones y agrupaciones.  
**Requisitos:**

* La construcción debe ser fluida y encadenada.
* Debe validar la sintaxis incrementalmente para evitar queries inválidos.
* Soporte para diferentes dialectos SQL (MySQL, PostgreSQL, Oracle).  
  **Funcionalidades:**
* Métodos para agregar selects, joins, where, group by, order by.
* Validación de que los nombres de tablas y columnas existen.
* Generación de queries parametrizadas para evitar SQL injection.

**7. Service Registry para Microservicios (Singleton)**

**Contexto:** Una arquitectura de microservicios necesita un service registry centralizado para descubrimiento dinámico de servicios y health checks.  
**Requisitos:**

* Solo debe existir una instancia del registry para evitar inconsistencia.
* Debe ser altamente concurrente y eficiente.
* Debe soportar failover y replicación para alta disponibilidad.  
  **Funcionalidades:**
* Registro y desregistro de servicios.
* Descubrimiento de servicios por nombre.
* Health checks periódicos para detectar servicios caídos.

**8. Factory de Estrategias de Enrutamiento para API Gateway (Factory Method)**

**Contexto:** Un API gateway necesita seleccionar estrategias de enrutamiento complejas basado en métricas en tiempo real y políticas de negocio.  
**Requisitos:**

* Las estrategias incluyen round-robin, latency-based, geolocation, etc.
* La decisión debe tomarse en runtime basado en múltiples factores.
* Debe ser fácil agregar nuevas estrategias.  
  **Funcionalidades:**
* Creación de estrategias basado en reglas configurables.
* Monitorización de métricas en tiempo real (latencia, carga).
* Fallback a estrategia default en caso de error.

**9. Clonación Profunda en Editor Gráfico Vectorial (Prototype)**

**Contexto:** Un editor gráfico vectorial necesita clonar elementos complejos (formas, grupos, capas) preservando referencias y dependencias.  
**Requisitos:**

* Los elementos tienen referencias cruzadas y anidadas.
* Las copias deben ser independientes pero estructuralmente idénticas.
* Alto rendimiento para no bloquear la UI al clonar elementos grandes.  
  **Funcionalidades:**
* Clonación profunda de gráficos vectoriales.
* Mantenimiento de referencias internas (ej: un grupo que contiene formas).
* Undo/redo para operaciones de clonación.

**10. Abstract Factory para Juego Multiplataforma (Abstract Factory)**

**Contexto:** Un juego debe ejecutarse en múltiples plataformas (PC, Consola, Mobile) con componentes gráficos, de sonido y de input específicos.  
**Requisitos:**

* Cada plataforma tiene su propia implementación de componentes.
* El código del juego debe ser independiente de la plataforma.
* Debe ser fácil portar el juego a nuevas plataformas.  
  **Funcionalidades:**
* Creación de familias de componentes por plataforma.
* Detección automática de la plataforma en runtime.
* Fallback graceful para componentes no soportados.

**11. Sistema de Reporting Cross-Platform (Abstract Factory)**

**Contexto:** Una aplicación de business intelligence debe generar reports en diferentes formatos (PDF, Excel, HTML) con componentes consistentes.  
**Requisitos:**

* Cada formato tiene sus propias implementaciones de tablas, gráficos y estilos.
* Los reports deben ser semanticamente equivalentes across formatos.
* Debe ser fácil agregar nuevos formatos de output.  
  **Funcionalidades:**
* Generación de reports en múltiples formatos desde la misma data.
* Mantenimiento de consistencia visual y semántica.
* Exportación simultánea a múltiples formatos.

**12. Circuit Breaker para Sistema Distribuido (Singleton)**

**Contexto:** Un sistema distribuido necesita monitorear el estado de servicios externos y evitar cascading failures.  
**Requisitos:**

* Solo una instancia por servicio monitoreado.
* Estado compartido entre todos los consumidores.
* Adaptación dinámica basado en historial de fallos.  
  **Funcionalidades:**
* Monitorización de estado de servicios externos.
* Apertura/cierre automático del circuito.
* Métricas de performance y logging de eventos.