Fácil

1.
Una aplicación de mensajería necesita que solo exista una única instancia del gestor de conexión al servidor durante toda la ejecución. Este gestor debe ser accesible globalmente, y garantizar que no se creen múltiples instancias accidentales en distintos módulos.

¿Cuál de las siguientes opciones describe la mejor solución usando un patrón de diseño GoF?

A. Crear la clase GestorConexion con un constructor público y una lista de instancias posibles.
B. Usar un método estático que devuelva siempre la misma instancia privada de GestorConexion.
C. Crear un gestor nuevo cada vez que una clase requiera conectarse al servidor.
D. Usar un patrón Observer para notificar a todas las instancias de conexión activas.

Respuesta correcta: B.
Justificación: Se aplica el patrón **Singleton**, que garantiza una única instancia global y controlada de una clase, proporcionando un punto de acceso centralizado.

2.
Un sistema de reservas de vuelos necesita construir objetos complejos como “Itinerario”, que contienen múltiples secciones (vuelos, hoteles, autos). La creación puede variar según el tipo de viaje (negocios, turismo, familiar), pero el proceso de construcción debe seguir una misma secuencia.

¿Cuál es la mejor opción de diseño?

A. Usar un patrón Builder que separe la construcción del objeto complejo de su representación final.
B. Crear subclases de Itinerario para cada tipo de viaje.
C. Utilizar un patrón Singleton para gestionar los distintos constructores.
D. Crear una factoría estática que devuelva itinerarios predefinidos.

Respuesta correcta: A.
Justificación: El patrón **Builder** permite construir objetos complejos paso a paso, con distintos ensamblajes, manteniendo el mismo proceso de construcción.

3.
Un sistema bancario necesita crear diferentes tipos de cuentas (ahorro, corriente, empresarial). Todas implementan la misma interfaz pero requieren diferentes procesos de inicialización. La aplicación cliente no debe depender de las clases concretas.

¿Qué patrón proporciona esta abstracción?

A. Prototype
B. Factory Method
C. Decorator
D. Proxy

Respuesta correcta: B.
Justificación: El patrón **Factory Method** delega la creación de objetos a subclases, permitiendo instanciar tipos concretos sin acoplar el código cliente.

4.
Un sistema gráfico necesita crear familias completas de objetos relacionados (Botón, Ventana, Menú) para distintas plataformas (Windows, macOS, Linux) sin modificar el código cliente.

¿Cuál es el patrón más adecuado?

A. Builder
B. Abstract Factory
C. Bridge
D. Prototype

Respuesta correcta: B.
Justificación: Abstract Factory permite crear familias de objetos relacionados sin especificar sus clases concretas, facilitando la portabilidad entre plataformas.

5.
Una aplicación de dibujo necesita clonar formas (círculos, rectángulos, líneas) que tienen configuraciones complejas (color, tamaño, textura). Copiar los valores manualmente sería ineficiente.

¿Qué patrón resuelve mejor este problema?

A. Prototype
B. Singleton

C. Builder
D. Adapter

Respuesta correcta: A.
Justificación: Prototype permite clonar objetos existentes evitando crear instancias desde cero, ideal para duplicar configuraciones complejas.

6.
En una aplicación de edición de texto, se desea añadir funcionalidades adicionales (como revisión ortográfica, conteo de palabras, formato automático) sin modificar la clase base del editor.

¿Qué patrón estructural es más adecuado?

A. Adapter
B. Decorator
C. Proxy
D. Facade

Respuesta correcta: B.
Justificación: El patrón **Decorator** permite agregar responsabilidades dinámicamente a un objeto sin alterar su estructura original.

7.
Un sistema multimedia necesita reproducir archivos con diferentes formatos. Las clases actuales solo aceptan formato MP3, pero se requiere compatibilidad con MP4 y WAV sin cambiar el código existente.

¿Qué patrón usarías?

A. Adapter
B. Facade
C. Proxy
D. Composite

Respuesta correcta: A.
Justificación: El **Adapter** convierte la interfaz de una clase en otra que el cliente espera, permitiendo la interoperabilidad entre componentes incompatibles.

8.
Un software complejo tiene múltiples subsistemas (facturación, inventario, usuarios). Se requiere una interfaz simple para que los clientes puedan realizar operaciones comunes sin conocer la estructura interna del sistema.

¿Qué patrón estructural aplicarías?

A. Facade
B. Bridge
C. Proxy
D. Composite

Respuesta correcta: A.
Justificación: Facade ofrece una interfaz unificada para un conjunto de interfaces de un subsistema, simplificando su uso.

9.
Un videojuego debe cambiar el comportamiento de su personaje principal según su estado (normal, herido, invencible). Cada estado altera la respuesta ante eventos como atacar o recibir daño.

¿Cuál patrón de comportamiento aplicarías?

A. Strategy
B. State
C. Observer
D. Chain of Responsibility

Respuesta correcta: B.
Justificación: El patrón **State** permite que un objeto modifique su comportamiento cuando cambia su estado interno, evitando condicionales complejos.

10.
Un sistema de monitoreo de sensores requiere que múltiples pantallas se actualicen automáticamente cuando cambien los valores de los sensores. Las pantallas no deben consultar activamente al sensor.

¿Qué patrón resuelve este requerimiento?

A. Command
B. Observer

C. Memento
D. Bridge

Respuesta correcta: B.
Justificación: Observer establece una relación de suscripción, notificando automáticamente a los observadores cuando cambia el estado del sujeto.

Medio

Están diseñadas para evaluar **comprensión profunda, diferencias entre patrones y su aplicación en contextos complejos**.

1.
Un sistema de análisis financiero debe generar diferentes reportes (PDF, HTML, Excel) usando el mismo proceso de construcción, pero los productos finales varían completamente en estructura y formato. Además, se requiere que los pasos de construcción sean controlados por el cliente, no por una jerarquía de fábricas.

¿Cuál patrón sería el más adecuado?

A. Abstract Factory
B. Builder
C. Factory Method
D. Prototype

Respuesta correcta: B.
Justificación: Builder permite al cliente controlar paso a paso la creación de un objeto complejo, separando el proceso de construcción de su representación final, a diferencia de Abstract Factory, que se enfoca en familias de productos completos.

2.
Una empresa de videojuegos desarrolla múltiples motores gráficos (OpenGL, DirectX, Vulkan). Se necesita mantener la lógica del juego independiente del motor, pero poder intercambiar el motor gráfico en tiempo de ejecución sin afectar el resto del sistema.

¿Qué patrón estructural se ajusta mejor?

A. Bridge
B. Adapter
C. Facade
D. Proxy

Respuesta correcta: A.
Justificación: Bridge desacopla la abstracción (lógica del juego) de su implementación (motor gráfico), permitiendo variar ambos independientemente y facilitar la extensibilidad.

3.
Una aplicación de mensajería permite enviar mensajes por diferentes medios (SMS, email, push). Se requiere crear un objeto "Notificador" que pueda combinar múltiples comportamientos dinámicamente (por ejemplo, notificar por email y SMS a la vez) sin crear clases combinadas por cada posible mezcla.

¿Qué patrón implementa mejor esta necesidad?

A. Composite
B. Decorator
C. Chain of Responsibility
D. Strategy

Respuesta correcta: B.
Justificación: Decorator permite agregar dinámicamente responsabilidades (métodos de envío adicionales) a un objeto sin crear combinaciones de clases estáticas.

4.
Un sistema de autenticación debe intentar validar a un usuario a través de varios métodos (contraseña, token, huella, reconocimiento facial) en orden, deteniéndose cuando uno tenga éxito.

¿Cuál patrón de comportamiento modela esta secuencia de responsabilidad?

A. State
B. Command
C. Chain of Responsibility
D. Strategy

Respuesta correcta: C.
Justificación: Chain of Responsibility pasa la solicitud a través de una cadena de manejadores hasta que uno la procesa, eliminando dependencias rígidas entre emisores y receptores.

5.
En una aplicación cliente-servidor, los objetos remotos deben parecer locales para el cliente, pero las llamadas deben interceptarse para añadir lógica de red, caché o seguridad.

¿Qué patrón estructural encapsula mejor este comportamiento?

A. Proxy
B. Adapter
C. Facade
D. Bridge

Respuesta correcta: A.
Justificación: Proxy actúa como intermediario, controlando el acceso al objeto real y permitiendo añadir funcionalidad adicional como control remoto, seguridad o lazy loading.

6.
En una aplicación de documentos colaborativos, varios observadores (usuarios conectados) deben recibir actualizaciones del documento compartido. Sin embargo, el sistema debe permitir conectar y desconectar observadores sin afectar al sujeto, y los observadores deben poder reaccionar de forma diferente al mismo evento.

¿Cuál patrón proporciona esta flexibilidad?

A. Observer
B. Mediator
C. Command
D. Memento

Respuesta correcta: A.
Justificación: Observer define una relación uno-a-muchos entre el sujeto y sus observadores, desacoplando ambos y permitiendo que cada observador actúe de forma independiente ante los cambios.

7.
Un editor gráfico mantiene el historial completo de modificaciones sobre un lienzo (añadir figura, cambiar color, mover objeto). Se necesita poder deshacer y rehacer cualquier acción sin acoplar el editor a las operaciones concretas.

¿Qué patrón implementa este historial de operaciones?

A. Memento
B. Command

C. Prototype
D. State

Respuesta correcta: B.
Justificación: Command encapsula cada operación (y su reversión) en un objeto, permitiendo almacenar, deshacer o rehacer acciones sin depender del editor concreto.

8.
En un entorno de simulación, el comportamiento de un objeto depende de su modo de operación (manual, automático, seguro). Las transiciones entre modos deben ser transparentes, sin condicionales extensos, y cada modo puede modificar cómo responden los mismos métodos.

¿Cuál patrón permite este cambio dinámico de comportamiento?

A. Strategy
B. State
C. Template Method
D. Bridge

Respuesta correcta: B.
Justificación: State encapsula los estados posibles de un objeto y permite que cambie su comportamiento dinámicamente según su estado actual, evitando múltiples condicionales.

9.
Una biblioteca de procesamiento de imágenes define un algoritmo general para filtrar imágenes, pero permite que las subclases redefinan pasos específicos como “ajustar brillo” o “aplicar máscara”, sin alterar la estructura del algoritmo completo.

¿Qué patrón de comportamiento aplica este principio?

A. Template Method
B. Strategy
C. Command
D. Observer

Respuesta correcta: A.
Justificación: Template Method define la estructura de un algoritmo en una clase base, delegando pasos específicos a subclases que pueden personalizar partes del proceso.

10.
Un sistema CAD necesita copiar objetos gráficos complejos (componentes con geometría, texturas y comportamientos) que podrían estar parcialmente configurados en memoria. Se desea crear nuevas instancias sin ejecutar todo el proceso de inicialización desde cero.

¿Cuál patrón es el más adecuado?

A. Builder
B. Prototype
C. Abstract Factory
D. Decorator

Respuesta correcta: B.
Justificación: Prototype permite clonar objetos existentes, incluso parcialmente configurados, evitando costos de inicialización y manteniendo independencia de las clases concretas.