# CLASE MENU

**1. Separación de responsabilidades**

Actualmente, la clase Menu maneja tanto la lógica de negocio como la interacción con el usuario (input/output). Esto la hace difícil de testear y mantener.

**Recomendación**: extraer la lógica relacionada con ingresos, gastos y validaciones a una clase separada (por ejemplo, ControladorFinanzas) y dejar Menú solo para manejar la entrada/salida.

**2. Evitar el uso de variables estáticas innecesarias**

La variable usuario y el scanner son static, lo que limita la flexibilidad del programa.

**Recomendación**: transformar la clase Menu en una instancia, pasando el Scanner como parámetro si es necesario. Esto permitiría instanciar Menu con diferentes entradas (ideal para testing).

**3. Validación de entradas del usuario**

Actualmente, se asume que el usuario siempre introduce valores válidos (como números). Esto puede causar errores en tiempo de ejecución.

**Recomendación**: validar las entradas del usuario con estructuras como try-catch o comprobaciones con scanner.hasNextDouble() antes de hacer nextDouble().

**4. Repetición de comprobaciones del usuario**

En varios métodos (introducirIngreso, introducirGasto, mostrarSaldo), se comprueba si el usuario está inicializado.

**Recomendación**: crear un método auxiliar como verificarUsuario() que centralice esta lógica y reduzca la repetición de código.

**5. Evitar múltiples responsabilidades en un método**

Por ejemplo, introducirGasto() hace demasiadas cosas: muestra opciones, obtiene el tipo de gasto, valida y procesa el gasto.

**Recomendación**: dividirlo en métodos más pequeños como elegirTipoGasto() y procesarGasto().

**6. Mejorar la experiencia del usuario**

Agregar mensajes más informativos, por ejemplo cuando hay un ingreso negativo, o si se intenta ingresar un gasto sin fondos suficientes.

**7. Mejora en el uso de constantes**

Los nombres de los tipos de gasto están en un switch.

**Recomendación**: utilizar una enum llamada TipoGasto que permita una mejor organización, validación y lectura del código.

Con estas mejoras, conseguiríamos lo siguiente:

- Código más limpio y legible.
- Facilita el testing con JUnit (al separar lógica de entrada/salida).
- Mejora la experiencia del usuario.
- Menor repetición de código.

# CLASE USUARIO

**1. Separar lógica de negocio y mensajes al usuario**

Actualmente, los métodos contienen llamadas a System.out.println(...), lo que mezcla la lógica de negocio con la presentación.

**Recomendación**: que los métodos devuelvan resultados (por ejemplo, mensajes o códigos de estado), y que sea la interfaz de usuario (como Menu) quien los imprima.

**2. Validaciones adicionales y consistentes**

El método ingresarSaldo valida que la cantidad sea positiva, pero registrarGasto no valida si la cantidad es mayor a 0.

**Recomendación**: añadir una validación similar en registrarGasto para evitar que se resten cantidades negativas.

**3. Evitar lógica innecesaria en el constructor**

Asignar saldo = 0.0; es redundante ya que Java lo hace automáticamente para double. No es grave, pero puede limpiarse para mantener el constructor más claro.

**4. Agregar métodos utilitarios si se usa desde otros sitios**

Por ejemplo, podrías añadir un método hasFondosSuficientes(double cantidad) si esta verificación se necesita en más de un lugar fuera de la clase.

**5. Preparar para la testabilidad**

Eliminar System.out directamente dentro de los métodos hace que estos sean más fáciles de testear.

**Recomendación**: retornar String con el mensaje o usar un patrón como Result si se busca algo más formal.

**6. Posibilidad de extender funcionalidad**

Actualmente, no se guarda el historial de ingresos o gastos.

**Opcional**: podrías añadir una lista de transacciones para permitir estadísticas, listados, etc.

Con estas mejoras, conseguiríamos lo siguiente:

- Mejora la separación de capas (negocio vs. presentación).
- Código más limpio y fácil de testear.
- Mayor consistencia en validaciones.
- Mayor extensibilidad para funcionalidades futuras (historial, reportes, etc.).