REVISIÓN DEL CÓDIGO – ACT. JUnit

REVISOR: David Gutérrez Ortiz

[https://github.com/DavidLazaro08/junitTesting.git]

PROYECTO EVALUADO: Carlos Madroñal Sánchez

[https://github.com/Carmasa/Simulador-Cajero.git]

1. VALORACIÓN GENERAL

El programa cumple perfectamente con el objetivo de gestionar ingresos y gastos de un usuario a través de un menú interactivo. La estructura está claramente dividida en clases como “Usuario”, “Cuenta”, “Gasto” e “Ingreso”, lo cual favorece el orden y que el código pueda leerse y entenderse sin problema alguno. Funciona correctamente en líneas generales y cubre los requisitos del enunciado, incluyendo además funcionalidades adicionales que van más allá como la opción de mostrar los ingresos ingresos y los gastos realizados, lo cual, pienso, mejora la experiencia del usuario.

La lógica principal está bien organizada, y desde el inicio del programa se recoge la información del usuario (nombre, y aunque no se pedía, DNI y fecha de nacimiento) antes de acceder al menú, lo cual da coherencia y continuidad a la aplicación. Por último, el sistema responde de forma estable cuando se añaden entradas erróneas (números en lugar de letras, etc) y permite interactuar con el usuario de manera fluida.

2. ENTORNO DE DESARROLLO Y PRUEBAS CON JUNIT

El proyecto fue adaptado a Maven, para gestionar bien dependencias y la ejecución de pruebas automáticas. Se ha podido importar y ejecutar sin complicaciones en IntelliJ. Las pruebas se realizaron en clases como Cuenta, Ingreso y Gasto. La clase “Usuario”, sin embargo, no pudo ser testeada correctamente debido a que sus métodos set contienen llamadas directas a Scanner (a través de la clase Depurar).

Esto significa que incluso si se le pasa un valor desde el test, el método lo ignora y vuelve a pedirlo por consola, lo cual interrumpe la automatización y bloquea la ejecución del test.3. Posibles mejoras detectadas.

REVISIÓN DEL CÓDIGO – ACT. JUnit

3. POSIBLES MEJORAS DETECTADAS

- En la clase “Usuario”, los setters piden los datos directamente con Scanner, lo que impide hacer pruebas automáticas. Sería mejor mover esa parte al menú a otra clase encargada de la entrada.
- Aunque las validaciones están bien gestionadas con la clase `Depurar`, añadir algún `try/catch` directamente en el menú ayudaría a separar mejor la lógica visual de la lógica de validación.
- Se podrían limitar los valores máximos que se pueden ingresar para evitar cantidades excesivas.
- Hay comentarios útiles en el código, pero en algunos métodos podrían detallarse un poco más para que todo sea más fácil de seguir.
- Se podría limitar la descripción de los ingresos y gastos a opciones fijas, como se indicaba.

4. CONCLUSIÓN

El programa funciona de forma estable y ofrece mejoras que van más allá del enunciado. Su estructura está bien pensada, y el uso de la clase Depurar permite gestionar bien errores de entrada. Solo cabría ajustar detalles como el uso de Scanner en los setters para facilitar las pruebas automáticas.