Tecnológico de Costa Rica

Curso:

Programación orientada a objetos

Documento:

Decisiones de desarrollo

Estudiantes:

Diego Castillo Fallas

David Fernández Torres

Josimar Spencer Suarez

Profesor

Luis Javier Chavarría Sánchez

Semestre

2

Año

2025

DOCUMENTO DE JUSTIFICACIÓN DE DECISIONES

Proyecto Programada – Remontando

1. Introducción

El presente documento pretende justificar las decisiones tomadas durante el diseño e implementación de la solución brindada por nuestras personas. Cada decisión fue analizada con base en los requerimientos del proyecto, los principios de la programación orientada a objetos, y el lineamiento de separación de responsabilidades (Separation of Concerns, SoC) mencionado en el documento brindado por el profesor.

2. Separación de Responsabilidades (SoC)

La estructura del proyecto se definió en tres paquetes:

- **model**: representa las entidades del dominio (Cliente, Instrumento y sus subclases).
- **util**: contiene la lógica de negocio (servicios), validaciones, ordenamiento, memoria y generación de códigos.
- **view**: maneja la interfaz con el usuario tanto en CLI como en GUI.

Justificación

Esta división permite:

- Aislar la lógica de negocio del modelo y la presentación.
- Facilitar el mantenimiento, prueba e independencia de cada capa.
- Cumplir estrictamente el principio SoC solicitado explícitamente en el documento del proyecto.

3. Jerarquía de Instrumentos

Las subclases Corriente, Pactada y Certificado heredan de la clase abstracta Instrumento.

Justificación

- Permite encapsular el comportamiento específico de cada instrumento.
- Habilita el polimorfismo, permitiendo tratar todos los instrumentos de forma uniforme.

- Facilita extender nuevos instrumentos sin modificar código existente.
- Cumple los principios de herencia, abstracción y cohesión solicitados en la rúbrica.

4. Tasas de interés, rangos y reglas de inversión

El documento del proyecto incluye tablas de interés con múltiples rangos, dependiendo de días o del saldo del instrumento.
Sin embargo, el ejemplo oficial de salida utiliza tasas anuales fijas por tipo de instrumento, y no aplica ninguna de las tablas.

Ejemplo oficial (PDF del profesor):

- Monto
- Plazo
- Una sola tasa anual
- Un cálculo directo y lineal

Decisión adoptada

Se implementaron tasas fijas por tipo de instrumento:

- Corriente: 2.00 %
- Pactada: 4.00 %
- Certificado: 6.00 %

Las tablas del documento se utilizaron como guía para diferenciar el comportamiento de cada instrumento (corriente, pactada y certificado), pero no se especifican de forma totalmente precisa reglas sobre:

- manejo de límites exactos de cada rango,
- combinación con tipo de moneda,
- comportamiento en plazos intermedios o superiores a los listados.

Definir una tasa base por tipo de instrumento permite:

- mantener el código simple y legible,
- obtener resultados consistentes y fácilmente verificables en CLI y GUI,

- concentrarse en la implementacion(modelo/servicios/vistas) sin introducir complejidad adicional en esta etapa.

La jerarquía de clases (Instrumento y sus subclases) deja el sistema preparado para incorporar las tablas completas más adelante. Bastaría con reemplazar la tasa base por lógica que consulte tablas de rangos o constantes predefinidas, sin modificar la estructura del diseño.

5. Manejo de memoria

Se creó la clase Memoria con listas estáticas para almacenar clientes e instrumentos.

Justificación

- Cumple el requerimiento del proyecto: “los datos deben manejarse en memoria principal”.
- Permite conservar los instrumentos aun si el cliente es eliminado.
- Evita dependencias externas, archivos o bases de datos.
- Simplifica los servicios y mantiene el sistema autocontenido.

6. Validación centralizada

Toda la validación se concentra en la clase Validador.

Justificación

- Evita duplicación de reglas y errores.
- Cumple la exigencia explícita de validar correos con regex.
- Refuerza el principio de SoC:
  el modelo no valida, la vista no decide reglas, el validador centraliza.

7. Manejo de excepciones

Las capas de servicio lanzan IllegalArgumentException, mientras que CLI y GUI las capturan.

Justificación

- Las vistas no deben aplicar reglas de negocio.
- Evita interrupciones de la ejecución.
- Mantiene el código claro y orientado a responsabilidades.

8. Ordenamiento

Se implementó Bubble Sort en Ordenamiento.

Justificación

- Es un algoritmo simple, legible y adecuado para listas pequeñas.
- Satisface la rúbrica sin introducir complejidad.
- Su implementación manual refuerza la comprensión de operaciones básicas sobre listas.

¿Qué es Bubble Sort?

- Bubble Sort es una forma muy simple de ordenar una lista.
  Funciona revisando la lista una y otra vez, comparando dos elementos que están juntos.
  Si están en el orden incorrecto, los intercambia.
- Repite este proceso hasta que todo esté ordenado.
- Es como cuando ordenás burbujas en agua:
  las más grandes van subiendo poco a poco hacia arriba, y las pequeñas quedan abajo.

9. Generación de códigos

GeneradorCodigo asigna códigos consecutivos tipo CLI-X.

Justificación

- Cumple con exactitud el formato solicitado.
- Asegura unicidad sin fuentes externas.
- Se integra naturalmente con la estructura del sistema.

10. Diseño de GUI

La clase GUIPrincipal administra paneles dinámicos mediante reemplazo en el panel central.

Justificación

- Permite una interfaz flexible, modular y ordenada.
- Evita mezclar múltiples funcionalidades en un solo panel.
- Mejora la mantenibilidad y la experiencia del usuario.
- Cumple estrictamente el principio de SoC.

11. Resolución de ambigüedades del documento

El enunciado no define por completo:

- cómo interpretar los rangos numéricos,
- cómo aplicar los límites,
- interacción de monedas,
- si las tablas son obligatorias o referenciales.

Decisión tomada

Usar tasas fijas, siguiendo el ejemplo oficial.

Justificación

- Garantiza que la salida del sistema coincida con la mostrada en el documento.
- Evita inconsistencias entre CLI y GUI.
- Mantiene la estructura simple, clara y alineada con los requerimientos pedagógicos del curso.

12. Conclusión

La solución desarrollada cumple con todos los requerimientos del proyecto, aplicando una arquitectura clara basada en separación de responsabilidades (SoC), un modelo orientado a objetos bien estructurado, validaciones centralizadas y un manejo adecuado de la interacción tanto en CLI como en GUI. Se diseñaron clases coherentes, se aplicó polimorfismo para los instrumentos financieros, se utilizó ordenamiento simple mediante Bubble Sort y se garantizó un manejo robusto de datos en memoria.

Además de resolver los requisitos funcionales, durante el desarrollo se aprendieron conceptos importantes como:

- la correcta separación entre modelo, lógica de negocio y vista,
- el uso de herencia y polimorfismo para evitar código repetido,
- la importancia de validar entradas y manejar excepciones,
- cómo construir una GUI modular con paneles reutilizables,
- y cómo mantener un diseño claro y fácil de mantener.

Esta experiencia permitió comprender de forma más profunda cómo estructurar un sistema completo desde cero, asegurando coherencia entre el diseño y la implementación