**Informe punto 3**

**Diseño de la solución(UML)**

A screenshot of a computer program

AI-generated content may be incorrect.

**Capturas de pantalla mostrando el funcionamiento de la calculadora, con ejemplos de cálculos básicos y avanzados.**

Menú principal, donde se encuentran todas la operaciones:  
A screenshot of a computer

AI-generated content may be incorrect.

Ejemplo de elegir una opción y hacer un calculo:

A black background with white text

AI-generated content may be incorrect.

A black background with white text

AI-generated content may be incorrect.

**Aplicación de los principios de la Programación Orientada a Objetos**

En el desarrollo de la Calculadora Científica en Kotlin, se aplicaron de manera práctica los principales principios de la Programación Orientada a Objetos (POO): encapsulamiento, herencia y polimorfismo. A continuación se describe cómo se implementa cada uno en el código.

**Encapsulamiento**

El encapsulamiento se utiliza para proteger los datos internos de las clases y controlar su acceso desde el exterior.  
En este programa, se aplica principalmente en:

* La clase CalculadoraCientifica, que mantiene el atributo memoria como privado (private var memoria: Double), evitando que sea modificado directamente por el usuario o por otras clases.  
  El acceso a este valor se controla mediante métodos públicos como:
  + memoriaAgregar(valor: Double)
  + memoriaRestar(valor: Double)
  + memoriaRecuperar()
  + memoriaLimpiar()

De esta forma, solo se puede modificar o leer el valor de la memoria a través de estos métodos, garantizando la integridad del dato.

* Además, dentro de la clase InterfazCalculadora, los métodos auxiliares como operacionSimple(), operacionDoble() y leerDouble() también están encapsulados (declarados como privados), ya que su uso está restringido al funcionamiento interno del menú interactivo.

El encapsulamiento, por tanto, asegura que cada clase controle su propio comportamiento interno y oculte los detalles innecesarios al resto del programa.

**Herencia**

La herencia se emplea para reutilizar código y extender funcionalidades sin duplicarlas.  
En este caso, la clase CalculadoraCientifica hereda de la clase base Calculadora mediante la sintaxis:

class CalculadoraCientifica : Calculadora()

Gracias a esto:

* La calculadora científica reutiliza las operaciones básicas (sumar, restar, multiplicar, dividir) definidas en Calculadora.
* A partir de esa base, amplía sus capacidades agregando nuevas funciones como seno, coseno, potencia, raizCuadrada, logaritmoNatural, evaluarExpresion, entre otras.

Este principio permitió construir una jerarquía lógica de clases donde la calculadora científica extiende el comportamiento de la calculadora básica sin necesidad de reescribir el código existente.

**Polimorfismo**

El polimorfismo se aplica cuando una clase derivada puede redefinir o sobrecargar comportamientos de la clase base, o cuando un mismo método puede comportarse de distintas formas según el tipo o número de parámetros.

En este proyecto, se presentan ambos tipos de polimorfismo:

1. Polimorfismo por sobrecarga de métodos (en la clase Calculadora):
   * Se definieron varios métodos con el mismo nombre pero con diferentes tipos de parámetros:
   * fun sumar(a: Double, b: Double): Double
   * fun sumar(a: Int, b: Int): Int
   * Esto permite que la misma operación (sumar, restar, etc.) funcione tanto con enteros como con números decimales.
2. Polimorfismo por herencia (o sobrescritura):
   * La clase CalculadoraCientifica podría sobrescribir métodos de la clase Calculadora si se deseara modificar su comportamiento, gracias a que los métodos base están marcados como open.
   * Aunque en este caso no se sobrescriben directamente, el diseño permite extender o personalizar el comportamiento de las operaciones básicas en futuras versiones sin alterar la clase base.

Gracias al polimorfismo, la calculadora puede manejar distintos tipos de datos y comportamientos de forma flexible y coherente.