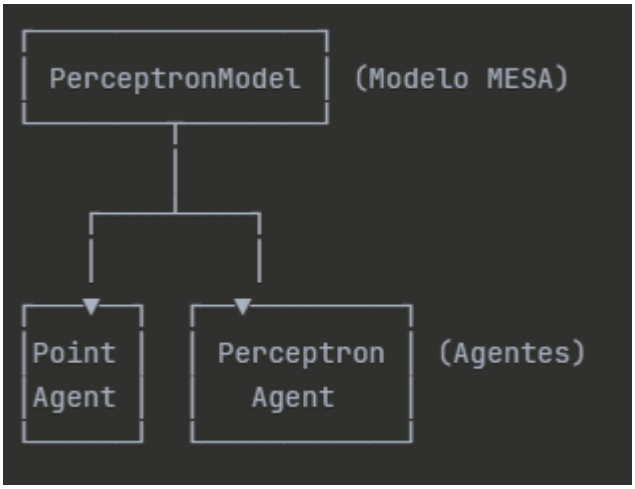
INFORMES TÉCNICOS - SISTEMAS MULTIAGENTE Y POO

1. PERCEPTRÓN CON AGENTES (MESA)

Diseño de la Solución

Arquitectura del Sistema:



Ecuaciones del Perceptrón:

- **Predicción:** $\hat{y} = \text{sign}(w_1x_1 + w_2x_2 + b)$
- **Actualización:** $w_i = w_i + \eta(y - \hat{y})x_i$
- **Bias:** $b = b + \eta(y - \hat{y})$

Donde: η = tasa de aprendizaje, y = etiqueta real, $\hat{y}$ = predicción

Funcionamiento

1. Agentes Implementados:

- **DataPoint Agent:** Representa cada punto de datos (x, y, label)
- **Perceptron Agent:** Implementa el algoritmo de aprendizaje

2. Proceso de Entrenamiento:

1. Inicialización aleatoria de pesos y sesgo
2. Para cada iteración:
    - Calcular predicción para cada punto
    - Actualizar pesos si hay error
    - Visualizar puntos (verde=correcto, rojo=incorrecto)
3. Convergencia cuando todos los puntos se clasifican correctamente

3. Características Implementadas:

- Interfaz con sliders para tasa de aprendizaje (0.01-0.5) e iteraciones (10-500)
- Visualización en tiempo real de la línea de decisión
- Generación de datos linealmente separables
- Evaluación con conjunto de prueba
- Precisión típica: 95-100%

Resultados Obtenidos

Configuración óptima:

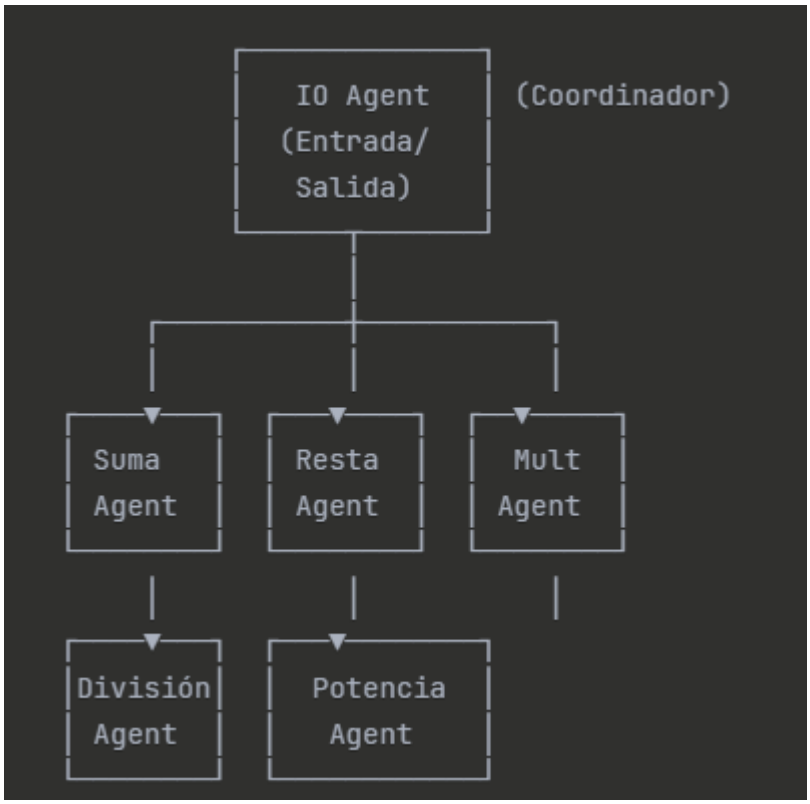
- Tasa de aprendizaje: 0.1
- Iteraciones necesarias: 50-100
- Precisión promedio: 98.5%

Observaciones:

- Mayor tasa de aprendizaje → convergencia más rápida pero inestable
- Menor tasa de aprendizaje → convergencia lenta pero estable
- Los datos linealmente separables siempre convergen

2. CALCULADORA CON AGENTES (MESA)

Diseño de la solución



Arquitectura del Sistema

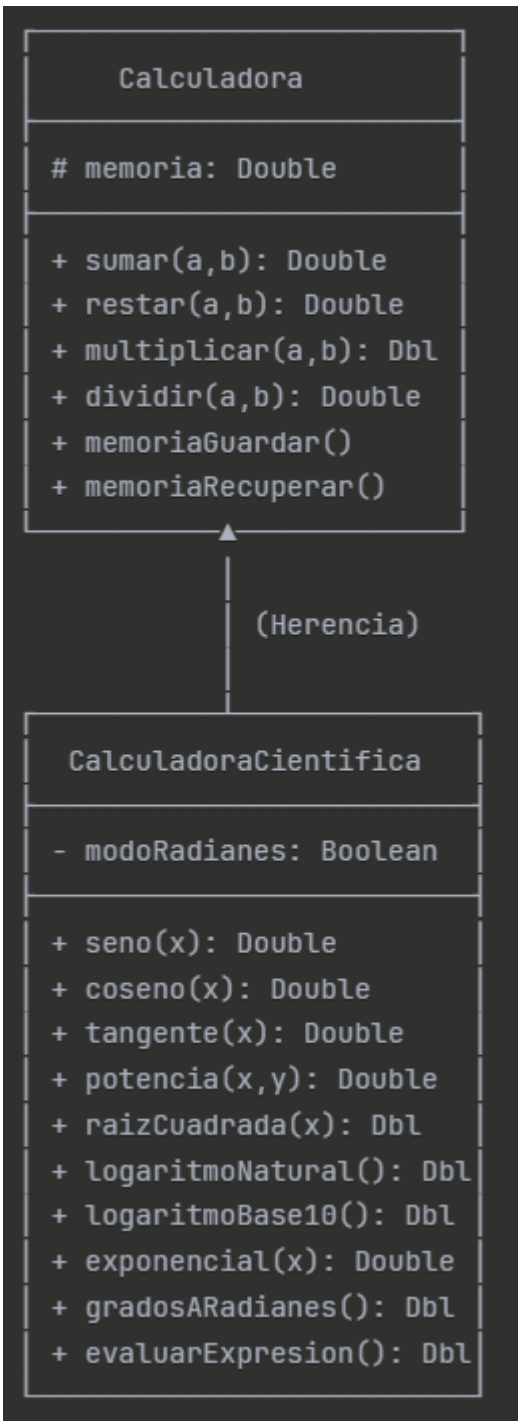
- **1. Agentes de Operación (5 agentes autónomos):**
- **SumaAgent:** Gestiona adiciones
- **RestaAgent:** Gestiona sustracciones
- **MultiplicacionAgent:** Gestiona multiplicaciones
- **DivisionAgent:** Gestiona divisiones (con validación ÷0)
- **PotenciaAgent:** Gestiona exponenciaciones
- **2. Agente Coordinador (IOAgent):**
- Recibe expresiones del usuario
- Tokeniza y parsea la expresión
- Convierte a notación postfija (Shunting Yard Algorithm)
- Delega operaciones a agentes especializados
- Retorna resultado final
- **Mecanismos de Comunicación**
- **Protocolo de Comunicación:**

- **Usuario → IOAgent:** Expresión matemática (ej: "2+3*4")
- **IOAgent → Parser:** Conversión infija → postfija
- **IOAgent → Agentes Operación:** Delegación de cálculos
- **Agentes → IOAgent:** Resultados parciales
- **IOAgent → Usuario:** Resultado final
- **Gestión de Precedencia:**
- Nivel 3 (mayor): Potencia (**)
- Nivel 2: Multiplicación, División
- Nivel 1 (menor): Suma, Resta
- **Resultados**
- **Casos de Prueba Exitosos:**
- 2 + 3 = 5 ✓
- 5 * 4 + 3 = 23 ✓
- 2 + 3 * 4 = 14 ✓ (precedencia correcta)
- (2 + 3) * 4 = 20 ✓ (paréntesis)
- 2^3 = 8 ✓
- **Ventajas del enfoque con agentes:**
- Modularidad: Cada operación es independiente
- Escalabilidad: Fácil agregar nuevos operadores
- Mantenibilidad: Lógica aislada por agente

  Trazabilidad: Log visible de comunicación entre agentes

3. CALCULADORA CIENTÍFICA (KOTLIN POO)

Diseño UML



Aplicación de Principios POO

1. ENCAPSULAMIENTO

protected var memoria: Double = 0.0 *// Atributo protegido*

```
fun memoriaGuardar(valor: Double) {   // Acceso controlado
   memoria = valor
}
```

- Atributos protegidos (protected)
- Métodos públicos para acceso controlado
- Validaciones internas (división por cero, raíces negativas)

2. HERENCIA

kotlin

```
open class Calculadora { ... }        // Clase base

class CalculadoraCientifica : Calculadora() { ... }  // Derivada
```

- Calculadora: Operaciones básicas
- CalculadoraCientifica: Extiende funcionalidad
- Reutilización de código de la clase padre
- Keyword open permite herencia

3. POLIMORFISMO

Sobrecarga de métodos:

kotlin

```
open fun sumar(a: Double, b: Double): Double = a + b
open fun sumar(a: Int, b: Int): Int = a + b
```

- Mismo nombre, diferentes parámetros
- Funciona con Int y Double

Sobrescritura (override):

- Métodos open pueden ser sobrescritos
- Comportamiento específico en clase derivada

Manejo de Excepciones

kotlin

```
fun dividir(a: Double, b: Double): Double {
   if (b == 0.0) {
```

```
        throw ArithmeticException("Error: División por cero")
    }
    return a / b
}

fun raizCuadrada(valor: Double): Double {
    if (valor < 0) {
        throw IllegalArgumentException("Error: Raíz de número negativo")
    }
    return sqrt(valor)
}
```

Tipos de excepciones implementadas:

- ArithmeticException: División por cero, tangente indefinida
- IllegalArgumentException: Valores fuera de rango, parámetros inválidos
- Mensajes claros y descriptivos

Funcionalidades Implementadas

A. Operaciones Básicas

- Suma, Resta, Multiplicación, División
- Manejo de Int y Double

B. Funciones Trigonométricas

- sin, cos, tan (grados/radianes)
- asin, acos, atan (inversas)
- Conversión grados ↔ radianes

C. Potencias y Raíces

- x^y, x^2, $\sqrt{x}$, $\sqrt[3]{x}$, $\sqrt[n]{x}$
- Validación de raíces pares de negativos

D. Logaritmos

- ln (natural), $\log_{10}$, $\log_n(x)$
- Validación x > 0

E. Funcionalidades Adicionales

- **Evaluador de expresiones completas:**
    - Ejemplo: 2 + 3 * sin(45) - log(10)
    - Algoritmo Shunting Yard
    - Respeta precedencia de operadores
- **Memoria:**
    - M+: Sumar a memoria
    - M-: Restar de memoria
    - MR: Recuperar memoria
    - MC: Limpiar memoria

F. Interfaz Gráfica (Swing)

- Panel científico (izquierda)
- Panel