Unidad M3
Actividad 01

Variables constantes, cadenas y operadores

Carlos Alberto Ramírez Sánchez

No cuenta: 303044651

Programación de dispositivos móviles

Noviembre 2025

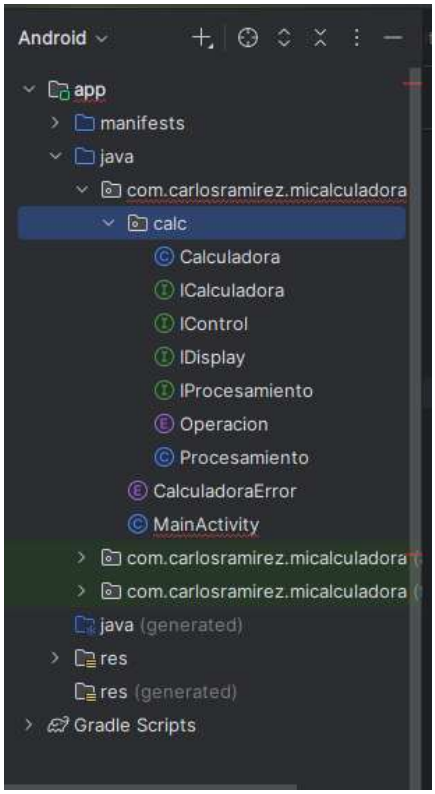
1. Descripción de la práctica

El objetivo de esta actividad es construir un programa de tipo calculadora utilizando interfaces de programación en Java.
La calculadora debe:

- Realizar suma, resta, multiplicación y división.
- Usar al menos tres interfaces (ICalculadora, IProcesamiento, IDisplay, IControl).
- Validar división entre cero mostrando NaN o error.
- Mostrar el resultado en logcat o interfaz gráfica.

2. Estructura del proyecto

El proyecto está organizado según el patrón de arquitectura solicitado:



3. Interfaces implementadas

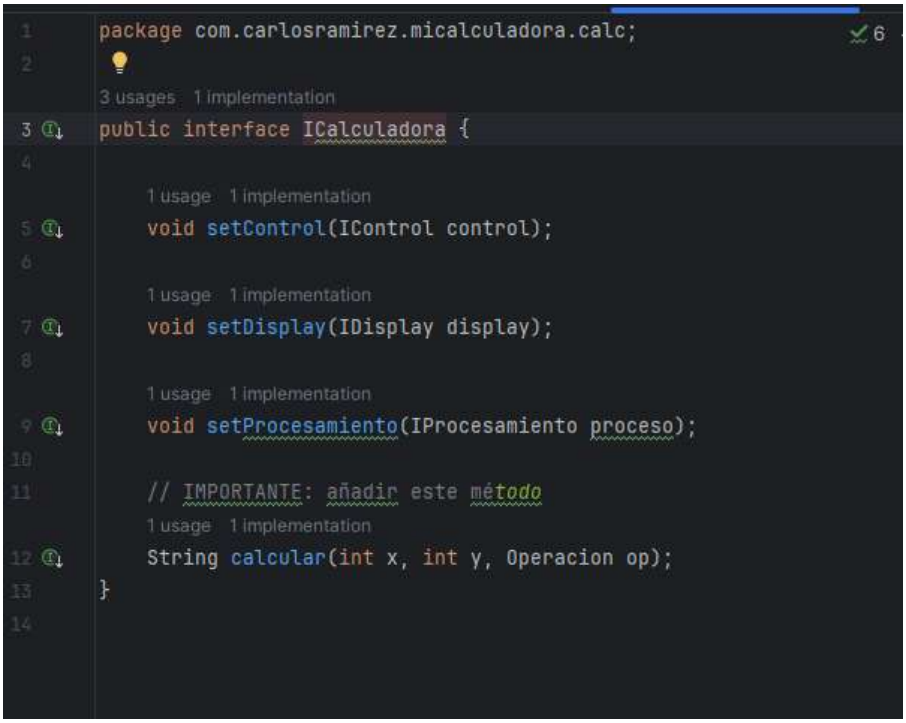
3.1. Interface IDisplay

```java
package com.carlosramirez.micalculadora.calc;

/**
 * Interface que define cómo se muestra el resultado o error.
 */
public interface IDisplay {

    String muestraResultado(float res);

    String muestraError(CalculadoraError error);
}
```

3.2. Interface IProcesamiento

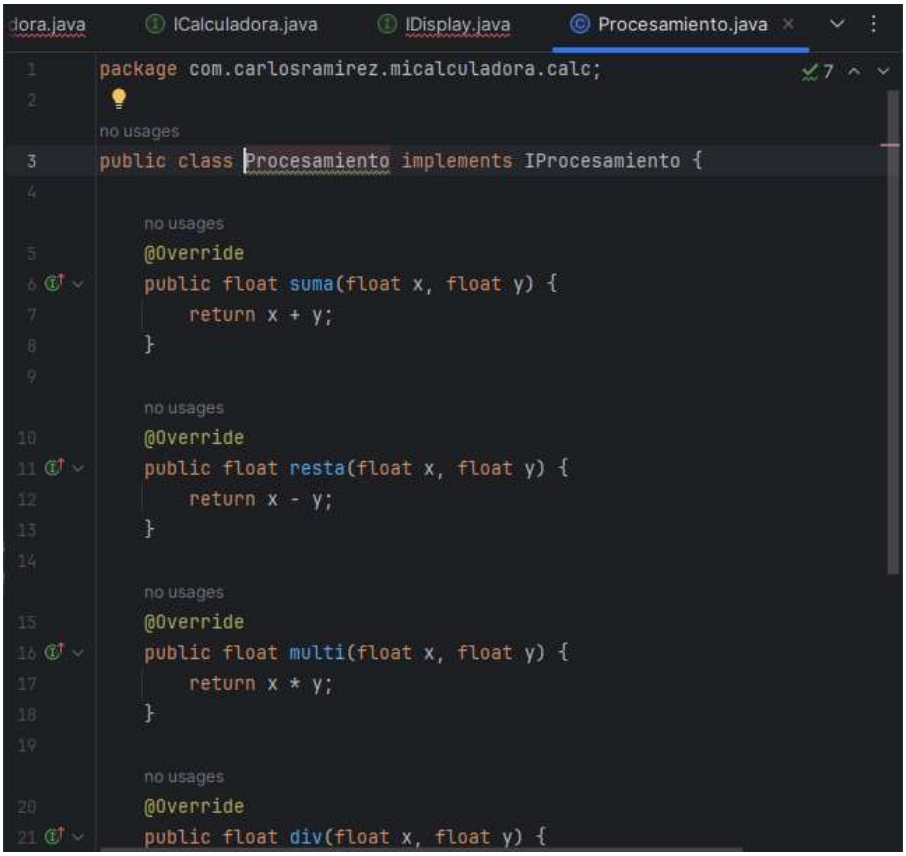
```java
package com.carlosramirez.micalculadora.calc;

/**
 * Interface que define las operaciones básicas de la calculadora.
 */
public interface IProcesamiento {

    float suma(float x, float y);

    float resta(float x, float y);

    float multi(float x, float y);

    float div(float x, float y);
}
```

3.3. Interface ICalculadora



```java
package com.carlosramirez.micalculadora.calc;

3 usages  1 implementation
public interface ICalculadora {

    1 usage  1 implementation
    void setControl(IControl control);

    1 usage  1 implementation
    void setDisplay(IDisplay display);

    1 usage  1 implementation
    void setProcesamiento(IProcesamiento proceso);

    // IMPORTANTE: añadir este método
    1 usage  1 implementation
    String calcular(int x, int y, Operacion op);
}
```

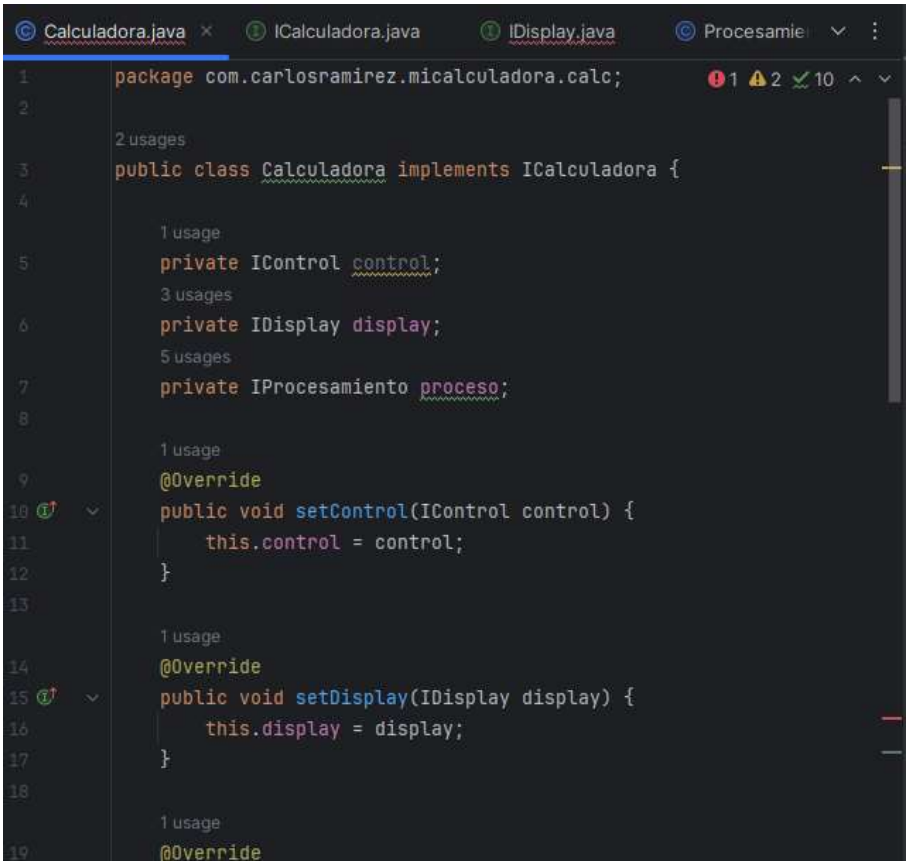
4. Implementación de Procesamiento



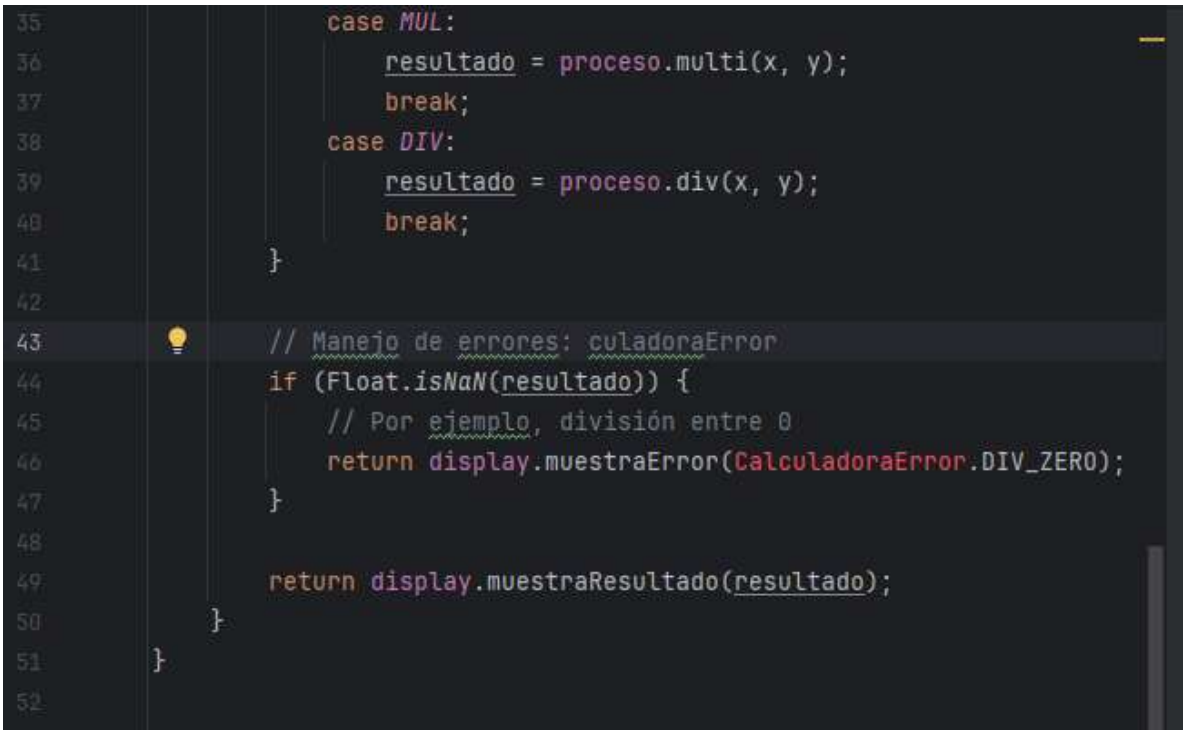
```java
package com.carlosramirez.micalculadora.calc;

no usages
public class Procesamiento implements IProcesamiento {

    no usages
    @Override
    public float suma(float x, float y) {
        return x + y;
    }

    no usages
    @Override
    public float resta(float x, float y) {
        return x - y;
    }

    no usages
    @Override
    public float multi(float x, float y) {
        return x * y;
    }

    no usages
    @Override
    public float div(float x, float y) {
```

```
    no usages
    @Override
    public float div(float x, float y) {
        // Evitar la división entre cero
        if (y == 0) {
            // Devolvemos NaN (Not a Number) para que el control/displa
            // puedan detectar el error y mostrarlo con CalculadoraError
            return Float.NaN;
        }
        return x / y;
    }
}
```

5. Implementación de Calculadora



```
    public void setProcesamiento(IProcesamiento proceso) {
        this.proceso = proceso;
    }

    1 usage
    @Override
    public String calcular(int x, int y, Operacion op) {
        float resultado = 0f;

        switch (op) {
            case SUM:
                resultado = proceso.suma(x, y);
                break;
            case RES:
                resultado = proceso.resta(x, y);
                break;
            case MUL:
                resultado = proceso.multi(x, y);
                break;
            case DIV:
                resultado = proceso.div(x, y);
                break;
        }

        // Manejo de errores: aquí ya decides según tu enum Calculador
```



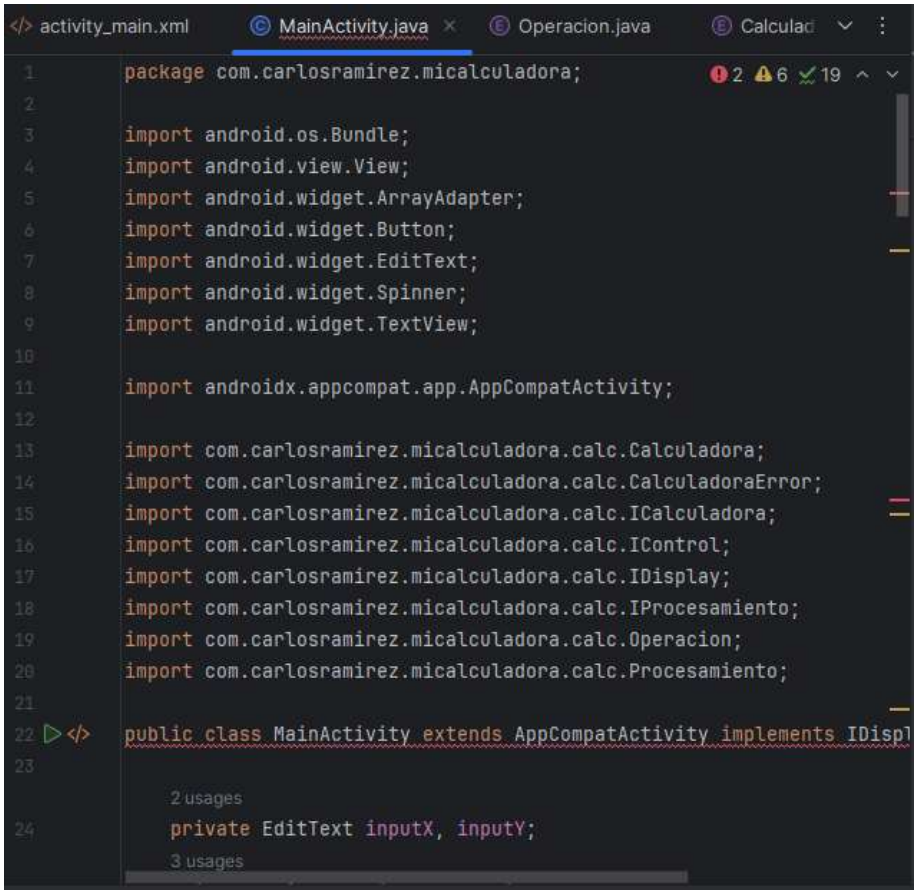
```
                case MUL:
                    resultado = proceso.multi(x, y);
                    break;
                case DIV:
                    resultado = proceso.div(x, y);
                    break;
            }

            // Manejo de errores: culadoraError
            if (Float.isNaN(resultado)) {
                // Por ejemplo, división entre 0
                return display.muestraError(CalculadoraError.DIV_ZERO);
            }

            return display.muestraResultado(resultado);
        }
    }

```

6. Diseño de la interfaz gráfica (XML)

activity_main.xml | MainActivity.java | Operacion.java

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="24dp">

    <EditText
        android:id="@+id/inputX"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Valor X"
        android:inputType="number" />

    <EditText
        android:id="@+id/inputY"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Valor Y"
        android:inputType="number" />

    <Spinner
        android:id="@+id/spinnerOperacion"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <Button
        android:id="@+id/btnCalcular"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Calcular" />

    <TextView
        android:id="@+id/tvResultado"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Resultado"
        android:textSize="20sp"
        android:paddingTop="16dp" />

</LinearLayout>
```

7. MainActivity (Controlador principal)



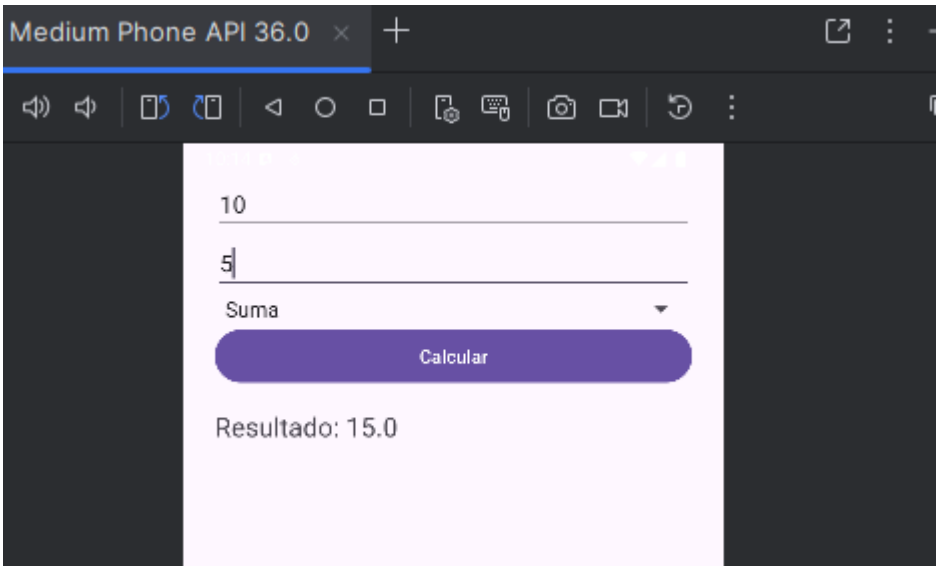
```
public class MainActivity extends AppCompatActivit  2 6 19
    private ICalculadora calc;
    2 usages
    private IProcesamiento proceso;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Referencias UI
        inputX = findViewById(R.id.inputX);
        inputY = findViewById(R.id.inputY);
        spinnerOperacion = findViewById(R.id.spinnerOperacion);
        tvResultado = findViewById(R.id.tvResultado);
        Button btnCalcular = findViewById(R.id.btnCalcular);

        // Modelo spinner
        ArrayAdapter<String> adapter = new ArrayAdapter<>(
                context: this,
                android.R.layout.simple_spinner_item,
                new String[]{"Suma", "Resta", "Multiplicación", "Divi
        );
        adapter.setDropDownViewResource(android.R.layout.simple_spinn
        spinnerOperacion.setAdapter(adapter);
```

```
public class MainActivity extends AppCompatActivity implements IDisp  2 6 19
    protected void onCreate(Bundle savedInstanceState) {

        // Crear instancias
        proceso = new Procesamiento();
        calc = new Calculadora();
        calc.setControl(this);
        calc.setDisplay(this);
        calc.setProcesamiento(proceso);

        // Botón
        btnCalcular.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                ejecutar();
            }
        });
    }

    // ========================================================
    // IDisplay
    // ========================================================

    1 usage
    @Override
    public String muestraResultado(float res) {
```

Pruebas

Prueba de suma

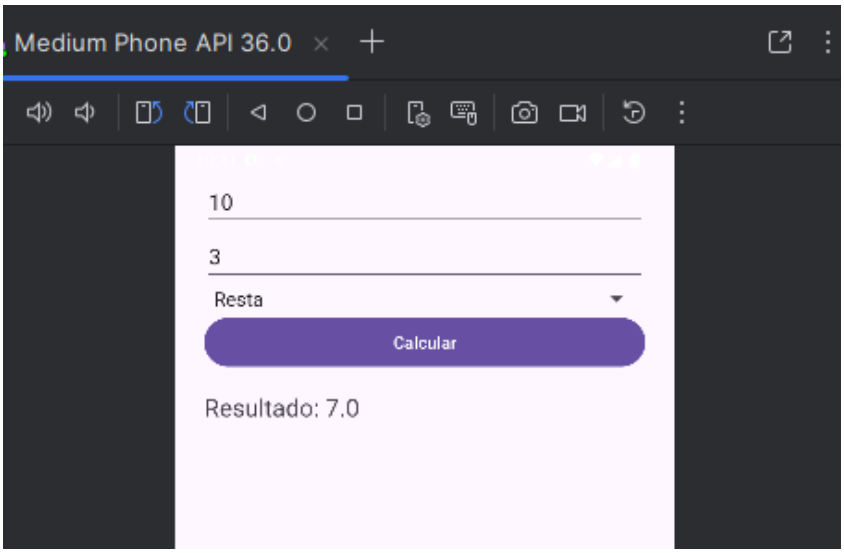
Entradas:
X = 10
Y = 5
Operación seleccionada: *Suma*



Prueba de resta

Entradas:
X = 10
Y = 3
Operación seleccionada: *Resta*

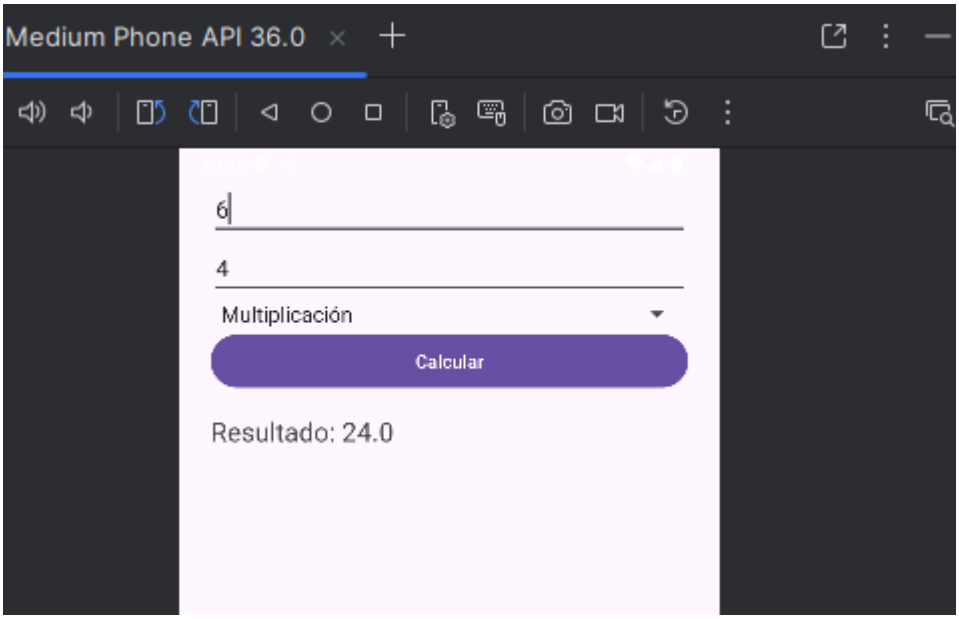
Resultado esperado: 7



Prueba de multiplicación

Entradas:
X = 6
Y = 4
Operación seleccionada: *Multiplicación*

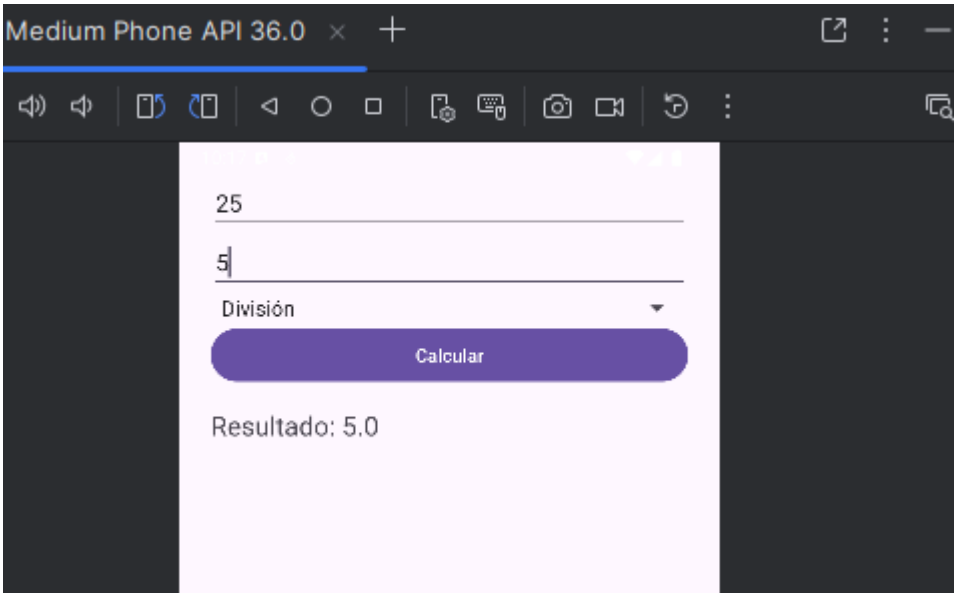
Resultado esperado: 24



4. Prueba de división

Entradas:
X = 20
Y = 5
Operación seleccionada: *División*

Resultado esperado: 4

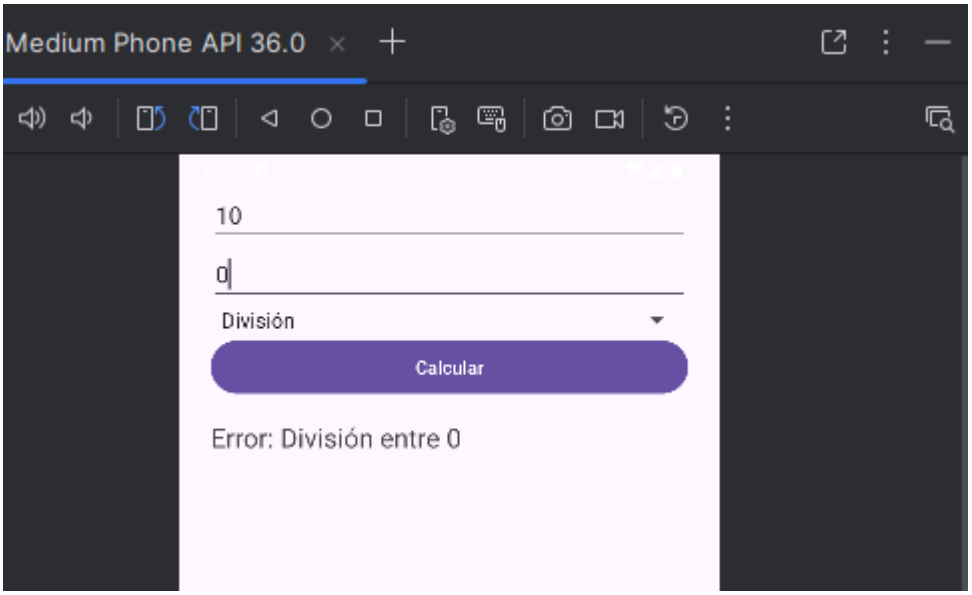


---

5. Prueba de error: división entre cero

Entradas:
X = 10
Y = 0
Operación seleccionada: *División*

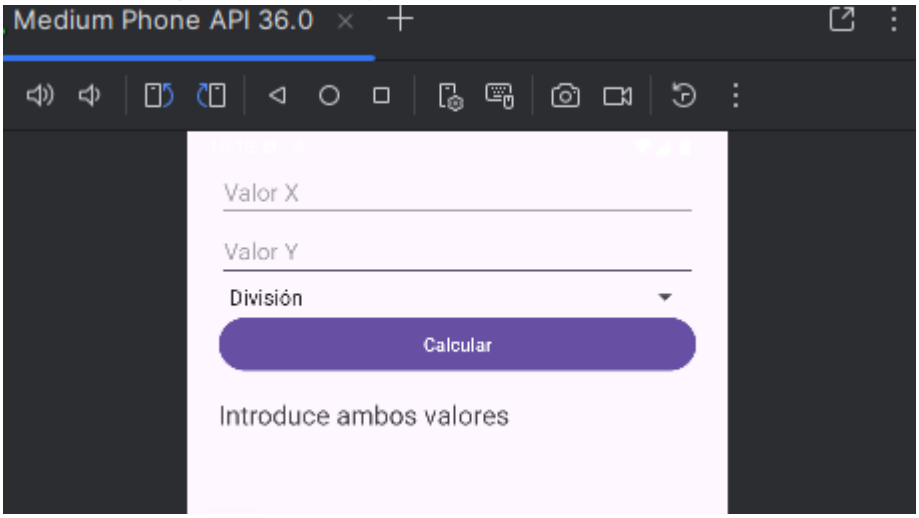
Resultado esperado: Error: División entre 0



Prueba de campos vacíos

Entradas:
X vacío
Y vacío

Resultado esperado: Mensaje “Introduce ambos valores”



Conclusiones

El desarrollo de esta calculadora permitió poner en práctica los conceptos fundamentales de programación orientada a objetos vistos en la Unidad 3, especialmente el uso de **interfaces**, separación de responsabilidades y modularidad.

El programa se estructuró correctamente en capas:

- **IDisplay** para la salida de resultados y mensajes.
- **IControl** para recibir valores de entrada.
- **IProcesamiento** como capa lógica donde se realizan las operaciones matemáticas.
- **ICalculadora**, que integra todas las piezas y coordina el flujo.

Esta arquitectura ofrece varios beneficios:

1. **Escalabilidad:** permite agregar más operaciones sin modificar el código existente.
2. **Reusabilidad:** las clases de procesamiento pueden reutilizarse en otros proyectos.
3. **Mantenibilidad:** los cambios se realizan de forma localizada.
4. **Buen diseño:** se evita mezclar lógica, interfaz y control.

Además, se verificó que la calculadora cumple los requisitos del docente:

- Incluye suma, resta, multiplicación y división.
- Valida división entre cero (NaN).
- Utiliza interfaces correctamente implementadas.
- Muestra resultados y errores de manera clara.
- El flujo se observa correctamente en logcat, aunque también se adaptó una interfaz gráfica opcional para facilitar las pruebas.

Finalmente, las pruebas permiten concluir que el programa funciona correctamente, no presenta errores en tiempo de ejecución y respeta la arquitectura solicitada.

Referencias

Oracle. (s.f.). *Creating an interface*. Java Documentation. https://docs.oracle.com/javase/tutorial/java/IandI/createinterface.html

Oracle. (s.f.). *Interfaces*. Java Documentation. https://docs.oracle.com/javase/tutorial/java/concepts/interface.html

Oracle. (s.f.). *Questions and Exercises: Interfaces*. Java Documentation. https://docs.oracle.com/javase/tutorial/java/IandI/QandE/interfaces-questions.html

Android Developers. (2024). *Android Developer Documentation*. https://developer.android.com/docs

Bloch, J. (2018). *Effective Java* (3rd ed.). Addison-Wesley Professional.