Universidad Nacional Autónoma de México

Facultad de Contaduría y Administración

Desarrollo de Aplicaciones móviles

Profesor: Cristian Cardoso Arellano

Alumno: Ortega Maldonado Diego Damiel

Actividad M3 01.

Interfaz de la calculadora.

```java
package com.example.m3_01;

//Interfaz para definir los métodos de la calculadora.
1 usage  1 implementation
public interface ICalculadora {

    no usages  1 implementation
    public Double suma(Double x, Double y);

    no usages  1 implementation
    public Double resta(Double x, Double y);

    no usages  1 implementation
    public Double multiplicacion(Double x, Double y);

    no usages  1 implementation
    public Double division(Double x, Double y);

}
```

Clase de implementación.

```java
package com.example.m3_01;

import android.util.Log;

//Clase de implementación de la interfaz.
1 usage
public class CalculadoraImpl implements ICalculadora{

    //Operadores.
    3 usages
    private Double x, y;

    //Sobrecargas.
    no usages
    @Override
    public Double suma(Double x, Double y) {
        return x + y;
    }

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

    no usages
    @Override
    public Double multiplicacion(Double x, Double y) {
        return x * y;
    }
```

```java
    @Override
    public Double division(Double x, Double y) {
        if(y == 0){

            Log.e( tag: "Calculadora",  msg: "Error NaN: divisón entre cero es indefinida.");
            return Double.NaN;

        }

        return x / y;

    }

    no usages
    public Double getX() {
        return x;
    }

    no usages
    public void setX(Double x) {
        this.x = x;
    }

    no usages
    public Double getY() {
        return y;
    }
```

```java
    no usages
    public void setY(Double y) {
        this.y = y;
    }

    //Constructor parametrizado.
    1 usage
    public CalculadoraImpl(Double x, Double y) {
        this.x = x;
        this.y = y;
    }

    //Constructor vacío.
    no usages
    public CalculadoraImpl(){

    }

}
```

Activar todo desde MainActivity.

```java
public class MainActivity extends AppCompatActivity {

    //Declarar como interfaz..
    5 usages
    private ICalculadora calculadora;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable( $this$enableEdgeToEdge: this);
        setContentView(R.layout.activity_main);
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
            Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
            return insets;
        });

        //Operandos con valores de prueba.
        Double x = 22.0;
        Double y = 4.0;

        //Instanciar la calculadora.
        calculadora = new CalculadoraImpl(x, y);
```

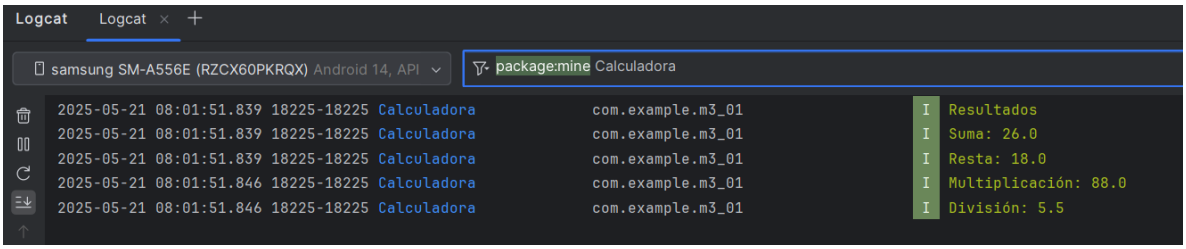
```java
        //Instanciar la calculadora.
        calculadora = new CalculadoraImpl(x, y);

        //Operar y mostrar en logcat.
        Log.i( tag: "Calculadora",  msg: "Resultados");
        Log.i( tag: "Calculadora",  msg: "Suma: " + calculadora.suma(x, y));
        Log.i( tag: "Calculadora",  msg: "Resta: " + calculadora.resta(x, y));
        Log.i( tag: "Calculadora",  msg: "Multiplicación: " + calculadora.multiplicacion(x, y));
        Log.i( tag: "Calculadora",  msg: "División: " + calculadora.division(x, y));

    }

}
```

Resultados en logcat.



Conclusión: El uso de interfaces para declarar el comportamiento esperado por una clase concreta es sin duda alguna una de las mayores ventajas de los lenguajes orientados a objetos, ya que al declarar únicamente el comportamiento de las

implementaciones se puede garantizar una separación de componentes y un desacoplamiento fuerte, sin mencionar que se cumple el principio I del acrónimo SOLID, la inversión de dependencias. Usar interfaces permite que al momento de implementarlas la lógica del negocio adicional sea responsabilidad de cada implementación, por lo que si se llega a necesitar otra lógica se puede reemplazar, pero el comportamiento base sigue ahí.