

USB

INGENIERIA DE SOFTWARE

# ALTA

COHESIÓN

JUAN RODRIGUEZ - DAVID VARGAS

2025

.....



```

other-app > src > JS App.js > App > render
7
8
9 getWeather = async () => {
10   const apiCall = fetch(`${apiBase}q=${this.state.city}`)
11   .then(res => res.json())
12   .then(res => {
13     console.log(res);
14     let city = res.name;
15     let country = res.sys.country;
16     let weatherDescription = res.weather[0].description;
17     let currentTemp = res.main.temp;
18     let maxTemp = res.main.temp_max;
19     let minTemp = res.main.temp_min;
20     }
21   console.log(weatherDescription);
22   console.log(currentTemp);
23   console.log(maxTemp);
24   console.log(minTemp);
25
26   this.setState({
27     city: city,
28     country: country,
29     weatherDescription: weatherDescription,
30     currentTemp: currentTemp,
31     maxTemp: maxTemp,
32     minTemp: minTemp
33   });
34 }

```

# ¿ALTA COHESIÓN?

LA COHESIÓN SE REFIERE A CUÁN ENFOCADA Y RELACIONADA ESTÁN LAS RESPONSABILIDADES DE UNA CLASE, MÓDULO O MÉTODO.

.....

ALTA COHESIÓN: UNA CLASE TIENE UN PROPÓSITO ÚNICO Y BIEN DEFINIDO

BAJA COHESIÓN: UNA CLASE INTENTA HACER DEMASIADAS COSAS NO RELACIONADAS



# EJEMPLO

# BAJA

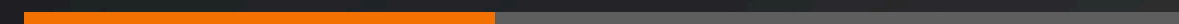
# COHESIÓN

## PROBLEMAS

- La clase Utilidades maneja operaciones matemáticas, envío de correos y acceso a bases de datos.
- No hay una relación clara entre sus métodos.
- Si hay un error en el envío de correos, afectará a toda la clase.
- Difícil de reutilizar (¿para qué usarías una clase que hace de todo?).

```
class Utilidades {  
    // Método para sumar dos números  
    public int sumar(int a, int b) {  
        return a + b;  
    }  
  
    // Método para enviar un correo  
    public void enviarCorreo(String destinatario, String mensaje) {  
        // Lógica de envío de correo...  
    }  
  
    // Método para guardar en una base de datos  
    public void guardarEnBD(String datos) {  
        // Lógica de base de datos...  
    }  
}
```

PERFORMANCE



REUSABILITY



EFFECTIVENESS



# EJEMPLO

# ALTA

# COHESIÓN

## VENTAJAS

- Cada clase tiene una responsabilidad única y clara.
- Si hay un error en ServicioCorreo, no afectará a las otras clases.
- Fácil de mantener, reutilizar y testear.

```
class Calculadora {  
    public int sumar(int a, int b) {  
        return a + b;  
    }  
  
    public int restar(int a, int b) {  
        return a - b;  
    }  
  
    public int multiplicar(int a, int b) {  
        return a * b;  
    }  
}  
  
class ServicioCorreo {  
    public void enviarCorreo(String destinatario, String mensaje) {  
        // Lógica de envío de correo...  
    }  
}  
  
class RepositorioDatos {  
    public void guardar(String datos) {  
        // Lógica de base de datos...  
    }  
}
```

PERFORMANCE



REUSABILITY



EFFECTIVENESS



# IMPORTANCIA

¿Por qué es Importante la Alta Cohesión?

M

## Mantenibilidad

Es más fácil corregir errores o añadir funcionalidades en clases enfocadas.

L

## Legibilidad

El código es más fácil de entender para otros desarrolladores (o para ti en el futuro).

R

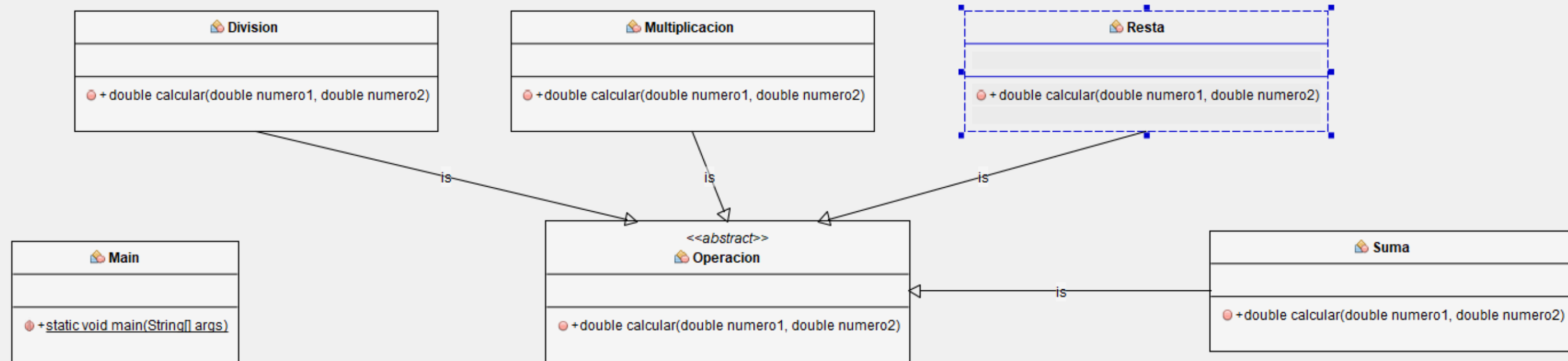
## Reutilización

Clases con propósitos claros pueden usarse en diferentes proyectos.

B

## Bajo acoplamiento

Las clases no dependen excesivamente de otras (otro principio clave de POO).



# EJEMPLO DE USO

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template
 */
package calculadora;

/**
 *
 * @author Juan
 */
public abstract class Operacion { // Clase Padre
    //Se fuerza a que las subclases implementen el método ya que tienen un logica comun
    public abstract double calcular(double numero1, double numero2); // Metodo que se heredara
}
```

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package calculadora;

/**
 *
 * @author Juan
 */
public class Resta extends Operacion { /////Hereda de Operacion y sobrescribe el método calcular() para sumar dos números.
    @Override //indica que estamos redefiniendo el método de la clase padre.
    public double calcular(double numero1, double numero2) {
        return numero1 - numero2;
    }
}
```

## Calculadora

### Source Packages

#### calculadora

- Calculadora.java
- Division.java
- Main.java
- Multiplicacion.java
- Operacion.java
- Resta.java
- Suma.java

---

**THANK YOU**

---