

**Tema:**

Calculadora en Java

**Alumno:**

Ronald Eduardo Mejia Reinosa

**Instructores:**

Oscar Lemus

Eduardo Calles

[**Documentación del programa 4**](#_48e3gf7vu6ul)

[Instalación y ejecución 4](#_423s9jiz6l7b)

[Clase Calculadora: 4](#_e96y9329wubp)

[Metodos: 4](#_g72878ovmxu2)

[Calculadora: 4](#_rfq1vwwgtto3)

[relizaOperacion: 4](#_2sqmkjqt8s16)

[Caja negra: 4](#_g0t46j17z44e)

[Entrada: 4](#_icyq6mt0bidh)

[Salida: 4](#_xcitcswwh2bd)

[Casos de prueba: 5](#_vi00tnwejxoa)

[Clase Operación: 5](#_16mtxnw5ob5a)

[Metodos: 5](#_ynkcgy1phqct)

[Operación: 5](#_luv1b2am66ad)

[getNombre: 5](#_93ucb1t7cmsl)

[setNombre: 5](#_ghocs7inii02)

[realizarOperacion: 5](#_j7ujemk0k72b)

[Caja negra: 5](#_sqvgf4uabz72)

[Entradas: 5](#_yobp2mjshbyi)

[Casos de prueba: 6](#_6s0vpilnx14f)

[Constructor 6](#_z7knimh3qb3)

[Método “IrealizarOperacion”: 6](#_vuizb2rbxe0a)

[Clase Número: 6](#_ln51znkblb6v)

[Metodos: 7](#_2vrt6wgv54p1)

[Número: 7](#_6zaqcbof5ruo)

[Número: 7](#_7cn3lc77fps)

[getValor: 7](#_ro8ubapfijbg)

[setValor: 7](#_biftw1nthyuk)

[Caja Negra 7](#_ij2g8rpfk5yf)

[Casos de prueba: 7](#_daiygaqn0ojh)

[Clase Suma: 9](#_w7jqb9b8spd8)

[Metodos: 9](#_o9fi1j2vlmm2)

[Suma: 9](#_g8za77dorr43)

[realizarOperacion: 9](#_nz7sfuknaekv)

[Caja negra: 9](#_6ie3rhsfynq6)

[Casos de prueba 10](#_n19l0bigm4ig)

[Clase Resta: 12](#_7xbxz6n5qs65)

[Metodos: 12](#_dgwmi3i9pj99)

[Resta: 12](#_zhf3niotr9mt)

[realizarOperacion: 12](#_xqgzi1tvphl6)

[Caja negra: 12](#_uf4wh632p82h)

[Casos de prueba 13](#_gkw45rxz5qts)

[Clase Multiplicación: 14](#_jm20ca63yo9a)

[Metodos: 15](#_vohcz5himim4)

[Multiplicacion: 15](#_ha8e51d7w7gv)

[realizarOperacion: 15](#_flknj9mxnamx)

[Caja negra: 15](#_jty8t6ypo3oo)

[Casos de prueba: 15](#_c4aixayrxe53)

[Clase División: 17](#_45y7ztnn6cg1)

[Metodos: 17](#_cvvopwjmy4iz)

[División: 17](#_355ixqv52i59)

[realizarOperacion: 18](#_n72wihe3rvs3)

[Caja negra: 18](#_x1wf3wdo7veo)

[Casos de prueba: 18](#_u6rnr0564ttl)

[Clase Potencia: 20](#_m8pzvu3lnn7q)

[Metodos: 20](#_g9471vphiszp)

[Potencia: 20](#_5a1d2ffspmlu)

[realizarOperacion: 20](#_w8iihatpfcab)

[Caja negra: 20](#_3ruh16xtb3vf)

[Casos de prueba 21](#_kv19hsht837q)

[Clase Raíz Cuadrada: 23](#_uoz2hlsisrv4)

[Metodos: 23](#_gig4in9cihx8)

[RaizCuadrada: 23](#_a1knvm6p3fma)

[realizarOperacion: 23](#_2umk5bb5yu7k)

[Caja negra: 24](#_v77vkfd3742g)

[Casos de prueba: 24](#_v4sem0nkoq4t)

[Clase Validaciones: 25](#_siev116z05v2)

[Metodos: 26](#_mjvu7anh8lo8)

[solicitarIntValido: 26](#_7bg2lp59mzal)

[pedirDoublePositivo: 26](#_3lgnftbjq4bj)

[Caja negra: 26](#_qjjuy8yyzxgz)

[Método solicitarIntValido 26](#_gzdqmhrmszq)

[Método pedirDoublePositivo 26](#_nxk2zj86f3qn)

[Casos de prueba: 27](#_d66mnikqbnpi)

# 

# Documentación del programa

## Instalación y ejecución

1. Clona el repositorio desde GitHub utilizando el comando

git clone https://github.com/REMR11/com.kodigo.calculadora.git

1. Abre el proyecto en IntelliJ IDEA y asegúrate de que tengas instalado el plugin de Maven.
2. Ejecuta el comando mvn clean package para compilar y empaquetar el proyecto.
3. Ejecuta el comando java -jar target/calculadora-1.0.jar para ejecutar la aplicación.

## Clase Calculadora:

Clase que representa una calculadora que puede realizar diferentes operaciones matemáticas. La calculadora utiliza una instancia de la clase Operación para realizar la operación específica.

### Metodos:

#### Calculadora**:**

Constructor que inicializa la calculadora con una operación específica. Recibe como parámetro a operación la operación que se va a realizar.

#### relizaOperacion:

Realiza la operación matemática específica sobre dos números. Recibe como parámetros a num1 el primer número, num2 el segundo número, devuelve el resultado de la operación. Atrapa errores si ocurre un error al realizar la operación.

### Caja negra:

#### Entrada:

**Operación:** una instancia de la clase Operación que se va a utilizar para realizar la operación matemática.

**num1 y num2:** los dos números que se van a utilizar para realizar la operación matemática.

#### Salida:

**resultado:** el resultado de la operación matemática realizada.

### Casos de prueba:

| Operación | Entrada | Salida |
| --- | --- | --- |
| Operación = new Suma() | num1 = 2  num2 = 3 | Resultado = 5 |
| Operación = new Resta() | num1 = 5  num2 = 2 | Resultado = 3 |
| Operación = new Multiplicar() | num1 = 4  num2 = 5 | Resultado = 20 |

## Clase Operación:

Clase abstracta que representa una operación matemática. Esta clase proporciona una implementación básica para las operaciones, incluyendo un nombre y métodos para obtener y establecer el nombre. Las clases que hereden de esta clase deben implementar el método abstracto realizarOperacion.

### Metodos:

#### **Operación:**

Constructor que inicializa la operación con su nombre. Recibe como parámetro a nombre el nombre de la operación. Atrapa errores si intentas ingresar un valor nulo.

#### **getNombre:**

Obtiene el nombre de la operación. Devuelve el nombre de la operación.

#### **setNombre:**

Establece el nombre de la operación. Recibe como parámetro a nombre el nuevo nombre de la operación.

#### **realizarOperacion:**

Realiza la operación matemática entre dos números. Debe ser implementado por las clases que hereden de esta clase. Recibe como parámetros a num1 el primer número y num2 el segundo número. Devuelve el resultado de la operación.

### Caja negra:

#### Entradas:

**Nombre:** El nombre de la operación (String).

**Num1 y Num2:** Los dos números que se van a utilizar para realizar la operación matemática (Clase Numero).

### Casos de prueba:

#### Constructor

| Entrada | Salida |
| --- | --- |
| nombre= “Suma” | Operación con nombre "Suma" |
| nombre= “Resta” | Operación con nombre "Resta" |
| nombre= “Multiplicación” | Operación con nombre "Multiplicación" |
| nombre= "División" | Operación con nombre "División" |

#### Método “realizarOperacion”:

| Entrada | Salida |
| --- | --- |
| num1 = 2  num2 = 3  Operacion con nombre "Suma | resultado = 5 |
| num1 = 5  num2 = 2  Operacion con nombre "Resta" | resultado = 3 |
| num1 = 4  num2 = 5  Operacion con nombre "Multiplicación" | resultado = 20 |
| num1 = 12  num2 = 3  Operacion con nombre "División" | resultado = 4 |
| num1 = 15  num2 = 0  Operacion con nombre "División" | ArithmeticException (no se puede dividir por cero) |

## Clase Número:

Clase que representa un número. Esta clase proporciona una implementación básica para un número, incluyendo un valor y métodos para obtener y establecer el valor.

### Metodos:

#### **Número:**

Constructor que acepta String como parámetro. Recibe como parámetro al valor que simboliza un número real.

#### **Número:**

Constructor que inicializa el número con su valor. Recibe como parámetro a valor el valor del número. Captura errores si ingresas un valor invalido.

#### **getValor:**

Obtiene el valor del número. Devuelve el valor del número.

#### **setValor:**

Establece el valor del número. Recibe valor el nuevo valor del número. Captura errores si ingresas un valor invalido.

### Caja Negra

**Entrada:**new Numero("10.5")

**Salida:** getValor() devuelve 10.5

**Entrada:** new Numero(Double.MAX\_VALUE)

**Salida:** getValor() devuelve Double.MAX\_VALUE

**Entrada:** new Numero("abc")

**Salida:** Lanza una IllegalArgumentException con el mensaje "El valor no es un número válido"

**Entrada:**

new Numero(Double.NaN)

**Salida:** Lanza una IllegalArgumentException con el mensaje "El valor no es un número válido"

### Casos de prueba:

| **n** | **Entrada** | **Salida esperada** | **Descripción** |
| --- | --- | --- | --- |
| 1 | new Numero("10.5") | getValor() devuelve 10.5 | Crear objeto Numero con valor válido |
| 2 | new Numero(Double.MAX\_VALUE) | getValor() devuelve Double.MAX\_VALUE | Crear objeto Numero con valor válido máximo |
| 3 | new Numero("abc") | Lanza IllegalArgumentException con mensaje "El valor no es un número válido" | Crear objeto Numero con valor inválido (no numérico) |
| 4 | Numero num = new Numero(5.0); num.setValor(10.0); | getValor() devuelve 10.0 | Asignar valor válido a objeto Numero existente |
| 5 | new Numero(Double.NaN) | Lanza IllegalArgumentException con mensaje "El valor no es un número válido" | Crear objeto Numero con valor inválido (NaN) |
| 6 | new Numero(Double.POSITIVE\_INFINITY) | Lanza IllegalArgumentException con mensaje "El valor no es un número válido" | Crear objeto Numero con valor inválido (infinito) |
| 7 | Numero num = new Numero(5.0); num.setValor(Double.NEGATIVE\_INFINITY); | Lanza IllegalArgumentException con mensaje "El valor no es un número válido" | Asignar valor inválido (infinito) a objeto Numero existente |

## Clase Suma:

Clase que representa la operación de suma. Esta clase se hereda de la clase abstracta Operación y proporciona una implementación para la operación de suma.

### Metodos:

#### **Suma:**

Constructor que inicia la operación de suma con su nombre.

#### **realizarOperacion:**

Realiza la operación de suma entre dos números. Recibe como parámetros a num1 el primer número y num2 el segundo número. Devuelve el resultado de la suma. Captura errores Si ocurre un error al realizar la suma

### Caja negra:

**Entrada:** new Suma().realizarOperacion(new Numero(5.0), new Numero(3.0))

**Salida:** 8.0

**Entrada:**

new Suma().realizarOperacion(new Numero(-5.0), new Numero(-3.0))

**Salida:** -8.0

**Entrada:** new Suma().realizarOperacion(new Numero(5.0), new Numero(-3.0))

**Salida:** 2.0

**Entrada:**

new Suma().realizarOperacion(new Numero(Double.MAX\_VALUE), new Numero(Double.MAX\_VALUE))

**Salida:**

Infinity (o un valor muy grande que supere el límite de Double.MAX\_VALUE)

**Entrada:**

new Suma().realizarOperacion(new Numero(5.0), new Numero(Double.NaN))

**Salida:** NaN

### Casos de prueba

| **Caso de prueba** | **Entrada** | **Salida esperada** | **Descripción** |
| --- | --- | --- | --- |
| 1 | Suma suma = new Suma(); suma.realizarOperacion(new Numero("5.0"), new Numero("3.0")) | 8.0 | Realizar suma con números positivos |
| 2 | Suma suma = new Suma(); suma.realizarOperacion(new Numero("-5.0"), new Numero("3.0")) | -2.0 | Realizar suma con números con signo diferente |
| 3 | Suma suma = new Suma(); suma.realizarOperacion(new Numero("5.0"), new Numero("-3.0")) | 2.0 | Realizar suma con números con signo diferente |
| 4 | Suma suma = new Suma(); suma.realizarOperacion(new Numero("0.0"), new Numero("0.0")) | 0.0 | Realizar suma con números cero |
| 5 | Suma suma = new Suma(); suma.realizarOperacion(new Numero("Double.MAX\_VALUE"), new Numero("1.0")) | Lanza RuntimeException con mensaje "Error al realizar la suma" | Realizar suma con número máximo y otro número |
| 6 | Suma suma = new Suma(); suma.realizarOperacion(new Numero("Double.NaN"), new Numero("1.0")) | Lanza RuntimeException con mensaje "Error al realizar la suma" | Realizar suma con número NaN y otro número |
| 7 | Suma suma = new Suma(); suma.realizarOperacion(new Numero("Double.POSITIVE\_INFINITY"), new Numero("1.0")) | Lanza RuntimeException con mensaje "Error al realizar la suma" | Realizar suma con número infinito y otro número |

## Clase Resta:

Clase que representa la operación de resta. Esta clase hereda de la clase abstracta Operación y proporciona una implementación para la operación de resta.

### Metodos:

#### **Resta:**

Constructor que inicializa la operación de resta con su nombre.

#### realizarOperacion:

Realiza la operación de resta entre dos números. Recibe como parámetros a num1 el primer número y num2 el segundo número. Devuelve el resultado de la resta. Captura errores si ocurre un error inesperado al realizar la resta.

### Caja negra:

**Entrada:** new Resta().realizarOperacion(new Numero(5.0), new Numero(3.0))

**Salida:** 2.0

**Entrada:** new Resta().realizarOperacion(new Numero(-5.0), new Numero(-3.0))

**Salida:** -2.0

**Entrada:** new Resta().realizarOperacion(new Numero(5.0), new Numero(-3.0)) **Salida:** 8.0

**Entrada:** new Resta().realizarOperacion(new Numero(Double.MAX\_VALUE), new Numero(Double.MAX\_VALUE))

**Salida:** 0.0 (o un valor muy pequeño)

**Entrada:** new Resta().realizarOperacion(new Numero(5.0), new Numero(Double.NaN))

**Salida:** NaN

**Entrada:** new Resta().realizarOperacion(new Numero(5.0), new Numero(Double.POSITIVE\_INFINITY))

**Salida:** -Infinity

**Entrada:** new Resta().realizarOperacion(null, null)

**Salida:** RuntimeException (o NullPointerException)

**Entrada:** new Resta().realizarOperacion(new Numero(5.0), null)

**Salida:** RuntimeException (o NullPointerException)

### Casos de prueba

| **Caso prueba** | **Entrada** | **Salida esperada** | **Descripción** |
| --- | --- | --- | --- |
| 1 | new Resta().realizarOperacion(new Numero(5.0), new Numero(3.0)) | 2.0 | Resta de dos números positivos |
| 2 | new Resta().realizarOperacion(new Numero(-5.0), new Numero(-3.0)) | -2.0 | Resta de dos números negativos |
| 3 | new Resta().realizarOperacion(new Numero(5.0), new Numero(-3.0)) | 8.0 | Resta de un número positivo y un número negativo |
| 4 | new Resta().realizarOperacion(new Numero(Double.MAX\_VALUE), new Numero(Double.MAX\_VALUE)) | 0.0 (o un valor muy pequeño) | Resta de dos números con valores máximos |
| 5 | new Resta().realizarOperacion(new Numero(5.0), new Numero(Double.NaN)) | NaN | Resta de un número y NaN |
| 6 | new Resta().realizarOperacion(new Numero(5.0), new Numero(Double.POSITIVE\_INFINITY)) | -Infinity | Resta de un número y Infinity |
| 7 | new Resta().realizarOperacion(null, null) | NullPointerException o RuntimeException | Resta de dos objetos Numero nulos |
| 8 | new Resta().realizarOperacion(new Numero(5.0), null) | NullPointerException o RuntimeException | Resta de un objeto Numero nulo y un objeto Numero no nulo |

## Clase Multiplicación:

Clase que representa la operación de multiplicación. Esta clase hereda de la clase abstracta Operación y proporciona una implementación para la operación de multiplicación.

### Metodos:

#### **Multiplicacion:**

Constructor que inicializa la operación de multiplicación con su nombre.

#### **realizarOperacion:**

Realiza la operación de multiplicación entre dos números. Recibe como parámetros a num1 el primer número y num2 el segundo número. Devuelve el resultado de la multiplicación. Captura errores si ocurre un error al realizar la multiplicación.

### Caja negra:

**Entrada:** new Multiplicacion().realizarOperacion(new Numero(5.0), new Numero(3.0))

**Salida:** 15.0

**Entrada:** new Multiplicacion().realizarOperacion(new Numero(-5.0), new Numero(-3.0))

**Salida:** 15.0

**Entrada:** new Multiplicacion().realizarOperacion(new Numero(5.0), new Numero(-3.0))

**Salida:** -15.0

**Entrada:** new Multiplicacion().realizarOperacion(new Numero(Double.MAX\_VALUE), new Numero(Double.MAX\_VALUE))

**Salida:** Infinity (o un valor muy grande)

**Entrada:** new Multiplicacion().realizarOperacion(new Numero(5.0), new Numero(Double.NaN))

**Salida:** NaN

### Casos de prueba:

| **Caso de prueba** | **Entrada** | **Salida esperada** | **Descripción** |
| --- | --- | --- | --- |
| 1 | new Multiplicacion().realizarOperacion(new Numero(5.0), new Numero(3.0)) | 15.0 | Multiplicación de dos números positivos |
| 2 | new Multiplicacion().realizarOperacion(new Numero(-5.0), new Numero(-3.0)) | 15.0 | Multiplicación de dos números negativos |
| 3 | new Multiplicacion().realizarOperacion(new Numero(5.0), new Numero(-3.0)) | -15.0 | Multiplicación de un número positivo y un número negativo |
| 4 | new Multiplicacion().realizarOperacion(new Numero(Double.MAX\_VALUE), new Numero(2.0)) | Infinity | Multiplicación de un número con valor máximo y un número finito |
| 5 | new Multiplicacion().realizarOperacion(new Numero(5.0), new Numero(Double.NaN)) | NaN | Multiplicación de un número y NaN |
| 6 | new Multiplicacion().realizarOperacion(new Numero(5.0), new Numero(Double.POSITIVE\_INFINITY)) | Infinity | Multiplicación de un número y Infinity |
| 7 | new Multiplicacion().realizarOperacion(null, null) | NullPointerException o RuntimeException | Multiplicación de dos objetos Numero nulos |
| 8 | new Multiplicacion().realizarOperacion(new Numero(5.0), null) | NullPointerException o RuntimeException | Multiplicación de un objeto Numero nulo y un objeto Numero no nulo |
| 9 | new Multiplicacion().realizarOperacion(new Numero(0.0), new Numero(5.0)) | 0.0 | Multiplicación de cero y un número finito |
| 10 | new Multiplicacion().realizarOperacion(new Numero(1.0), new Numero(5.0)) | 5.0 | Multiplicación de uno y un número finito |

## Clase División:

Clase que representa la operación de división. Esta clase hereda de la clase Operación y proporciona la implementación específica para la división.

### Metodos:

#### División:

Constructor que inicializa la operación de división con su nombre.

#### realizarOperacion:

Realiza la operación de división entre dos números. Recibe como parámetros a num1 el dividendo y num2 el divisor. Devuelve el resultado de la división. Captura errores si se intenta dividir por cero.

### Caja negra:

**Entrada:** new Division().realizarOperacion(new Numero(10.0), new Numero(2.0))

**Salida:** 5.0

**Entrada:** new Division().realizarOperacion(new Numero(-10.0), new Numero(-2.0))

**Salida:** 5.0

**Entrada:** new Division().realizarOperacion(new Numero(10.0), new Numero(-2.0))

**Salida:** -5.0

**Entrada:** new Division().realizarOperacion(new Numero(10.0), new Numero(0.0))

**Salida:** ArithmeticException: No se puede dividir por cero

**Entrada:** new Division().realizarOperacion(new Numero(10.0), new Numero(Double.POSITIVE\_INFINITY))

**Salida:** 0.0

**Entrada:** new Division().realizarOperacion(new Numero(10.0), new Numero(Double.NaN))

**Salida:** NaN

#### Casos de prueba:

| Caso de prueba | Entrada | Salida esperada | Descripción |
| --- | --- | --- | --- |
| 1 | new Division().realizarOperacion(new Numero(10.0), new Numero(2.0)) | 5.0 | División de dos números positivos |
| 2 | new Division().realizarOperacion(new Numero(-10.0), new Numero(-2.0)) | 5.0 | División de dos números negativos |
| 3 | new Division().realizarOperacion(new Numero(10.0), new Numero(-2.0)) | -5.0 | División de un número positivo y un número negativo |
| 4 | new Division().realizarOperacion(new Numero(10.0), new Numero(0.0)) | ArithmeticException: No se puede dividir por cero | División de un número por cero |
| 5 | new Division().realizarOperacion(new Numero(10.0), new Numero(Double.POSITIVE\_INFINITY)) | 0.0 | División de un número por Infinity |
| 6 | new Division().realizarOperacion(new Numero(10.0), new Numero(Double.NaN)) | NaN | División de un número por NaN |
| 7 | new Division().realizarOperacion(new Numero(0.0), new Numero(10.0)) | 0.0 | División de cero por un número finito |
| 8 | new Division().realizarOperacion(new Numero(1.0), new Numero(10.0)) | 0.1 | División de uno por un número finito |
| 9 | new Division().realizarOperacion(null, null) | NullPointerException o RuntimeException | División de dos objetos Numero nulos |
| 10 | new Division().realizarOperacion(new Numero(10.0), null) | NullPointerException o RuntimeException | División de un objeto Numero nulo y un objeto Numero no nulo |

#### 

## Clase Potencia:

Clase que representa la operación de potencia. Esta clase hereda de la clase abstracta Operación y proporciona una implementación para la operación de potencia.

### Metodos:

#### **Potencia:**

Constructor que inicializa la operación de potencia con su nombre.

#### **realizarOperacion:**

Realiza la operación de potencia sobre dos números. Recibe como parámetros a num1 la base de la potencia y num2 el exponente de la potencia. Devuelve el resultado de la potencia. Captura errores si ingresas un parámetro negativo o un cero.

### Caja negra:

**Entrada:** new Potencia().realizarOperacion(new Numero(2.0), new Numero(3.0))

**Salida:** 8.0

**Entrada:** new Potencia().realizarOperacion(new Numero(-2.0), new Numero(3.0))

**Salida:** -8.0

**Entrada:** new Potencia().realizarOperacion(new Numero(2.0), new Numero(-3.0))

**Salida:** 0.125

**Entrada:** new Potencia().realizarOperacion(new Numero(0.0), new Numero(3.0))

**Salida:** 0.0

**Entrada:** new Potencia().realizarOperacion(new Numero(0.0), new Numero(-3.0)) **Salida:** Infinity

**Entrada:** new Potencia().realizarOperacion(new Numero(2.0), new Numero(Double.NaN))

**Salida:** NaN

**Entrada:** new Potencia().realizarOperacion(new Numero(2.0), new Numero(Double.POSITIVE\_INFINITY))

**Salida:** Infinity

**Entrada:** new Potencia().realizarOperacion(null, new Numero(3.0))

**Salida:** NullPointerException o RuntimeException

**Entrada:** new Potencia().realizarOperacion(new Numero(2.0), null)

**Salida:** NullPointerException o RuntimeException

**Entrada:** new Potencia().realizarOperacion(null, null)

**Salida:** NullPointerException o RuntimeException

### Casos de prueba

| **Caso de prueba** | **Entrada** | **Salida esperada** | **Descripción** |
| --- | --- | --- | --- |
| 1 | new Potencia().realizarOperacion(new Numero(2.0), new Numero(3.0)) | 8.0 | Potencia de un número positivo con un exponente positivo |
| 2 | new Potencia().realizarOperacion(new Numero(-2.0), new Numero(3.0)) | -8.0 | Potencia de un número negativo con un exponente positivo |
| 3 | new Potencia().realizarOperacion(new Numero(2.0), new Numero(-3.0)) | 0.125 | Potencia de un número positivo con un exponente negativo |
| 4 | new Potencia().realizarOperacion(new Numero(0.0), new Numero(3.0)) | 0.0 | Potencia de un número cero con un exponente positivo |
| 5 | new Potencia().realizarOperacion(new Numero(0.0), new Numero(-3.0)) | Infinity | Potencia de un número cero con un exponente negativo |
| 6 | new Potencia().realizarOperacion(new Numero(2.0), new Numero(Double.NaN)) | NaN | Potencia de un número con un exponente NaN |
| 7 | new Potencia().realizarOperacion(new Numero(2.0), new Numero(Double.POSITIVE\_INFINITY)) | Infinity | Potencia de un número con un exponente Infinity |
| 8 | new Potencia().realizarOperacion(null, new Numero(3.0)) | NullPointerException o RuntimeException | Potencia de un número nulo con un exponente |
| 9 | new Potencia().realizarOperacion(new Numero(2.0), null) | NullPointerException o RuntimeException | Potencia de un número con un exponente nulo |
| 10 | new Potencia().realizarOperacion(null, null) | NullPointerException o RuntimeException | Potencia de dos números nulos |

## **Clase Raíz Cuadrada:**

Clase que representa la operación de raíz cuadrada. Esta clase hereda de la clase abstracta Operación y proporciona una implementación para la operación de raíz cuadrada.

### Metodos:

#### RaizCuadrada:

Constructor que inicializa la operación de raíz cuadrada con su nombre.

#### realizarOperacion:

Realiza la operación de raíz cuadrada sobre un número. Recibe como parametros a num1 el número sobre el que se realiza la raíz cuadrada y num2 no se utiliza en esta operación. Devuelve el resultado de la raíz cuadrada. Captura errores si ingresas un valor negativo o cero.

### Caja negra:

**Entrada:** new RaizCuadrada().realizarOperacion(new Numero(16.0), new Numero(0.0))

**Salida:** 4.0

**Entrada:** new RaizCuadrada().realizarOperacion(new Numero(0.0), new Numero(0.0))

**Salida:** 0.0

**Entrada:** new RaizCuadrada().realizarOperacion(new Numero(-16.0), new Numero(0.0))

**Salida:** ArithmeticException: No se puede calcular la raíz cuadrada de un número negativo

**Entrada:** new RaizCuadrada().realizarOperacion(null, new Numero(0.0))

**Salida:** NullPointerException

**Entrada:** new RaizCuadrada().realizarOperacion(new Numero(16.0), null)

**Salida:** NullPointerException

### Casos de prueba:

| **Caso de prueba** | **Entrada** | **Salida esperada** | **Descripción** |
| --- | --- | --- | --- |
| 1 | new RaizCuadrada().realizarOperacion(new Numero(16.0), new Numero(0.0)) | 4.0 | Raíz cuadrada de un número positivo |
| 2 | new RaizCuadrada().realizarOperacion(new Numero(0.0), new Numero(0.0)) | 0.0 | Raíz cuadrada de un número cero |
| 3 | new RaizCuadrada().realizarOperacion(new Numero(-16.0), new Numero(0.0)) | ArithmeticException | Raíz cuadrada de un número negativo |
| 4 | new RaizCuadrada().realizarOperacion(null, new Numero(0.0)) | NullPointerException | Raíz cuadrada con un número nulo |
| 5 | new RaizCuadrada().realizarOperacion(new Numero(16.0), null) | NullPointerException | Raíz cuadrada con un exponente nulo |
| 6 | new RaizCuadrada().realizarOperacion(new Numero(Double.NaN), new Numero(0.0)) | NaN | Raíz cuadrada de un número NaN |
| 7 | new RaizCuadrada().realizarOperacion(new Numero(Double.POSITIVE\_INFINITY), new Numero(0.0)) | Infinity | Raíz cuadrada de un número Infinity |

## **Clase Validaciones:**

Clase que proporciona métodos para realizar validaciones de entrada de usuario. Los métodos de esta clase se encargan de solicitar al usuario que ingrese valores numéricos válidos y de validar la entrada para asegurarse de que se cumplan las condiciones específicas

### Metodos:

#### solicitarIntValido:

Método encargado de validar si el número ingresado cumple como un entero y no avanzara hasta obtener un número válido. Recibe como parámetro a scanner que es utilizado para obtener información de consola. Devuelve int número válido. Captura errores si ingresas cualquier carácter que no sea un dígito..

#### **pedirDoublePositivo:**

Solicita al usuario que ingrese un valor doble positivo y valida la entrada. Recibe como parámetro a scanner que es utilizado para obtener información de consola. Devuelve un valor doble positivo válido. Captura errores si ingresas cualquier carácter que no sea un dígito.

### Caja negra:

#### Método solicitarIntValido

**Entrada:** 42 (un número entero positivo)

**Salida:** 42 (el mismo número entero positivo)

**Entrada:** -10 (un número entero negativo)

**Salida:** El número debe ser positivo. (mensaje de error) y se solicita nuevamente la entrada

**Entrada:** abc (un carácter no numérico)

**Salida:** Entrada no válida. Por favor ingrese un número decimal válido. (mensaje de error) y se solicita nuevamente la entrada

#### Método pedirDoublePositivo

**Entrada:** 3.14 (un número doble positivo)

**Salida:** 3.14 (el mismo número doble positivo)

**Entrada:** -0.5 (un número doble negativo)

**Salida:** Debe ser un número positivo. (mensaje de error) y se solicita nuevamente la entrada

**Entrada:** abc (un caracter no numérico)

**Salida:** Debe ser un número decimal válido. (mensaje de error) y se solicita nuevamente la entrada

### Casos de prueba:

Método solicitarIntValido

| **Caso de prueba** | **Entrada** | **Salida esperada** | **Descripción** |
| --- | --- | --- | --- |
| 1 | 42 | 42 | Número entero positivo |
| 2 | -10 | El número debe ser positivo. | Número entero negativo |
| 3 | abc | Entrada no válida. Por favor ingrese un número decimal válido. | Caracter no numérico |
| 4 | 0 | El número debe ser positivo. | Número entero cero |
| 5 | 123 | 123 | Número entero positivo grande |

Método pedirDoublePositivo

| **Caso de prueba** | **Entrada** | **Salida esperada** | **Descripción** |
| --- | --- | --- | --- |
| 1 | 3.14 | 3.14 | Número doble positivo |
| 2 | -0.5 | Debe ser un número positivo. | Número doble negativo |
| 3 | abc | Debe ser un número decimal válido. | Caracter no numérico |
| 4 | 0.0 | Debe ser un número positivo. | Número doble cero |
| 5 | 123.45 | 123.45 | Número doble positivo grande |

..