# Clase Math.

# ¿Qué es la clase Math?

*Math es una clase predefinida en java que viene en el paquete java.lang que se importa automaticamente en las clases java, por lo tanto, no requiere de una instancia tal como lo hace la clase Scanner que requiere de java.util. La clase Math solo requiere que se llamen desde la propia clase.*

*Algunos de los metodos de la clase Math son:*

1. ***Math.abs(x):*** *Devuelve el valor absoluto de la variable (x)*
2. ***Math.signum(x):*** *Devuelve el sigo de la variable (x)*
3. ***Math.pow(x, y):*** *evalua un base (x) al exponente (y) que se ingresa*
4. ***Math.sqrt(x):*** *Devuelve la raíz caudrada de la variable (x)*
5. ***Math.cbrt(x):*** *Devuelve la raíz cúbica de la variable (x)*
6. ***Math.hypot(x, y):*** *Básicamente el teormea de Pitágoras $\sqrt{(x^2 + y^2)}$*
7. ***Math.round(x):*** *Devuelve el redondeo simple de la variable (x)*
8. ***Math.ceil(x):*** *Devuelve el redondeo superior de la variable(x)*
9. ***Math.floor(x):*** *Devuelve el redonndeo inferior de la variable (x)*
10. ***Math.nextUp(x):*** *Devuelve el float/double superior más cercano*
11. ***Math.nextDown(x):*** *Devuelve el float/double inferior más cercano*
12. ***Math.max(x, y, ...):*** *Devuelve el valor máximo entre las variables ingresadas*
13. ***Math.min(x, y, ...):****Devuelve el valor mínimo entre las variables ingresadas*
14. ***Math.random():*** *Este método genera un numero aleatorio entre 0.0 y 0.1, pero, si se multiplica por una variable x (ej. 4) generara un valor entre 0 y ese número de variables (si es 4 sería de 0-3); y si se le suma 1 esta generara un numero aleatorio entre 1 y x (si es 4 sería de 1-4)*
15. ***Math.sin(x):*** *Evaluara el Sin de la variable (x) en radianes*
16. ***Math.cos(x):*** *Evaluara el Cos de la variable (x) en radianes*
17. ***Math.cos(x):*** *Evaluara la Tan de la variable (x) en radianes*
18. ***Math.toRadians(x):*** *Convierte la variable (x) a radianes*
19. ***Math.toDegrees(x):*** *Convierte la variable (x) a Grados*
20. ***Math.log(x):*** *Evalúa el logaritmo natural de la variable (x)*
21. ***Math.log10(x):*** *Evalúa el logaritmo base 10 de la variable (x)*
22. ***Math.exp(x):*** *Evalúa Euler a la x ($e^x$)*

***Ejemplos del funcionamiento en el .java adjunto con este archivo***

*Algunas de constantes de la clase Math:*

1. ***Math.PI**: El valor de π (aproximadamente 3.14159).*
2. ***Math.E**: El valor de la constante de Euler (aproximadamente 2.71828).*
3. ***Math.LN2**: El logaritmo natural de 2 (aproximadamente 0.693147).*
4. ***Math.LN10**: El logaritmo natural de 10 (aproximadamente 2.302585).*
5. ***Math.LOG2E**: El logaritmo en base 2 de e (aproximadamente 1.442695).*
6. ***Math.LOG10E**: El logaritmo en base 10 de e (aproximadamente 0.434294).*
7. ***Math.TAU**: El valor de 2π (aproximadamente 6.283185)*

## *¿Cómo generar un número random?*

*Existen varias formas de generar un número random aparte de Math.random() una de ellas es Random que requiere una instancia llamada java.util.Random,*
*Al igual que Scanner necesita definirse con*
*Random random = new Random()*
*Y su forma de ejecutarse seria*
*int NumRand = random.nextInt(n);*
*donde n es el rango que desea, Tambien puede randomizar con decimales usando double NumRand = random.nextDouble();*
*Otra forma es utilizando SecureRandom que es muy similar a Random en su fucionalidad*
*java.util.SecureRandom*
*SecureRandom securerandom = new SecureRandom();*
*Int NumRand = secureRandom.nextInt(n)*

---

*Nota: Los métodos nextUp y nextDown funcionan con variables double y float; Los métodos max y min no funcionan con arreglos directos; Las constantes pueden incluirse en los métodos, ej: Math.sin(Math.PI/2)=1.*

***Importante:** En el archivo PPT.java que es el que contiene el código del juego piedra, papel o tijeras, hay un método llamado try catch del que investigue por mi parte al varias veces confudirme y escribir texto en lugar de un número, quise que en lugar de que se crasheara por ingresar un valor no esperado lo analizara y volviera a preguntar.*

## Investigación aparte try-catch.

Método try-catch.
El método funciona como:

```
try {
        int a = scanner.nextInt();
        return;
} catch (InputMismatchException e) {
        System.out.println("Error!! \nIngresa un numero");
        scanner.nextLine();
} finally {
        String A=scanner.nextLine();
}
```

try lleva lo que debería de ocurrir de manera de normal en caso de que haya un error ese inmediatamente detendrá el resto y lo mandara a catch, es como un continue hacia catch con la condición de "sii error".
catch funcionará con la excepción que hayamos impuesto, manejará y controlará el error generado en try.
finally no es una línea necesaria todas ocasiones, esta se ejecutará independientemente si hay en try hay una excepción o no.

Tipos de Excepciones:

1) **Excepciones Comprobadas**. Son aquellas excepciones que el compilador obliga a manejar de alguna manera
   a) IOException: Se usa cuando hay un problema con el flujo de datos sea de salida o entrada como leer o escribir un archivo.
   b) SQLException: Se utiliza cuando hay error relacionado a la base de datos ya sea una conexión o una consulta fallida.
   c) ClassNotFoundException: Se lanza cuando intentas cargar una clase mediante su nombre (usando Class.forName()) y la clase no se encuentra en el classpath. **Nota: no entendí este así que solo lo copié**
   d) FileNotFoundException: Una subclase de IOException que se genera cuando intentas acceder a un archivo inexistente
   e) ParseException: Un error de un parcero XD. Se genera cuando a una cadena tatas de darle un formato y este falla, como con fechas, ej: 31\13\2025 → dd/MM/yyyy.
   f) InterruptedException: Esta excepción se lanza cuando un hilo se ve interrumpido mientras está esperando, durmiendo o realizando alguna operación de entrada/salida.
   g) NoSuchMethodException: Ocurre cuando se intenta acceder a un método que no existe.

2) **Excepciones no Comprobadas**. Generalmente errores de lógica del programa
   a) NullPointerException: Se genera cuando haces intentas hacer referencia a un objeto vacío (null).
   b) ArrayIndexOutOfBoundsException: Sucede cuando apuntas fuera del rango de un arreglo.
   c) ArithmeticException: Sucede cuando surge un error aritmético como dividir entre x/0 o sacar √-1
   d) ClassCastException: Surge cuando quieres intentas realizar una conversión de tipo incorrecto con tipos incompatibles.

e) IllegalArgumentException: Surge cuando un método recibe un argumento inválido o inapropiado.
f) IllegalStateException: Se lanza cuando un método es llamado en un objeto en un estado inapropiado. **Nota: Tampoco entendí esta excepción .-.,**
g) NumberFormatException: Surge cuando se intenta convertir una cadena en un número y el formato no es correcto.
h) UnsupportedOperationException: Surge cuando intentas ejecutar una operación que no soporta la clase o interfaz.

3) **Errores (Error).** Estos a diferencia de los otros son errores que no pueden ser manejados dentro de código.
   a) OutofMemoryError: Surge cuando no hay memoria suficiente para coninuar con la ejecución del programa.
   b) StackOverflowError: Surge cuando se desborda la pila de ejecución, por lo regular un ciclo recursivo sin una salida correcta.
   c) NoClassDefFoundError: Ocurre cuando java complica una clase, pero al momento de ejecutarse no la encuentra.
   d) LinkageError: Aparece cuando surge un error en la vinculación dinámica de clases, suele ocurrir cuando las clases no son compatibles o si las versiones de las clases son incompatibles

4) **InputMismatchException**: Es uno que requiere una instancia para utilizarse (java.util.InputMismatchException) y este surge cuando se ingresa un tipo de dato diferente al esperado, ej. Un double, string o char cuando se espera un int. En mi caso este es el error que utilice.
5) **NoSuchElementException**: Es muy similar al anterior con la excepción de que este surge cuando ya no hay más datos que leer ya sea un objeto o un dato ingresado, también requiere de una instancia para utilizarse (java.util.NoSuchElementException).
6) **Puedes crear una exception**: Utilizando una super clase

```
public class ValidacionException extends Exception {
    public ValidacionException(String mensaje) {
        super(mensaje);
    }
}
public class Main {
    public static void main(String[] args) {
        try {
            throw new ValidacionException("¡Error de validación! El valor es incorrecto.");
        } catch (ValidacionException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

7) **Hay más, pero son los que más me interesaron.**

Notas: **1)** Se pueden encadenar varios catch para un mismo try, lo cual es útil cuando se trabaja con algo con muchos errores. **2)** Es recomendable utilizar y manejar excepciones de manera específica un ArithmeticException para métodos con ejecuciones aritméticas, esto para evitar evasiones de errores. **3)** A partir de java 7 se pueden manejar varias excepciones en un solo bloque como si de condicionales se tratara:

```
try{
} Catch (InputMismatchException || NoSuchElementException) {
}
```

. **4)** Una superclase maneja un dato de una clase (como me lo explico con peras y manzanas XD) ej. La superclase será Fruta y tenemos una manzana con un sabor especifico y una pera con sabor especifico que serán las subclases, ambas son frutas, pero de diferentes sabores, lo que hacen las subclases es llamar al constructor de la superclase Fruta para guardar el nombre de la fruta y su respectivo sabor; y puede ser utilizado por estas en cualquier momento. **5)** Para crear una excepción tiene que definirse que tipo es, “extends Exception” para excepciones comprobadas y “extends RuntimeException” para excepciones no comprobadas, estas últimas no requieren ser manejadas explícitamente.

**6)** Y yo que solo utilice try-catch con una excepción especifica._.’