

JAVA HANDBOOK



Escrito por: María Fernanda López Ojeda

¿QUÉ ES JAVA?

Java es una plataforma informática y un tipo de lenguaje de programación orientado a objetos que permite la creación de aplicaciones de forma sencilla.



Fue diseñado por la compañía Sun Microsystems Inc con el fin de construir un lenguaje capaz de funcionar en redes computacionales heterogéneas (redes formadas por más de un tipo de computadora o sistema operativo ya sea PC, MAC's, estaciones de trabajo, etc), además de ser independiente de la plataforma donde se va a ejecutar para ocuparlo en cualquier máquina

PROGRAMACIÓN ORIENTADA A OBJETOS

La programación orientada a objetos (POO) es la base de Java. De hecho, todos los programas de Java están por lo menos a un cierto grado orientados a objetos. Este método de programación intenta resolver problemas representados en un programa informático modelando los objetos del mundo real.

OBJETO

Se puede definir como una entidad (real o abstracta) con un papel definido en el dominio del problema. Java ejecuta sus datos como objetos y crea interfaces para esos objetos.



Un objeto contiene:

- Estado: Conjunto de propiedades y valores actuales de esas propiedades. Representado por el contenido de sus atributos. Los atributos son los valores o características de los objetos.
- Comportamiento: Modo en que el objeto actúa y reacciona, son los cambios de su estado y paso de mensajes. Se representa por métodos de un objeto. Además está determinado por la clase a la que pertenece el objeto y refleja su respuesta con otros objetos.

- Identidad: Propiedad que distingue a un objeto de otros, le da un nombre único de variable a un objeto y permite que interactúe con otros objetos.

Cada objeto en la realidad tiene propiedades propias de sí mismo (estado), como un avión tiene tamaño, capacidad, color, peso, etc.

Además, cada objeto puede realizar una serie de acciones (comportamiento). El avión puede despegar, aterrizar o moverse. Todas estas acciones se describen con verbos.

A las propiedades mencionadas anteriormente se les llama atributos, a las acciones métodos, y al conjunto de ambos características.



SUBIR



BAJAR



ABRIR



GIRAR

Un objeto es algo que se ve, se utiliza y juega un papel o un rol en el dominio de un problema del programa.

CLASE

Es el conjunto de objetos con estados y comportamientos similares.

Una clase es una plantilla que establece las variables y los métodos comunes para todos los objetos de un cierto tipo. Una clase es la aplicación de un tipo abstracto de datos y describe los atributos (estado) del objeto y sus operaciones (comportamiento).

ATRIBUTOS



VARIABLES

- **VELOCIDAD**
- **ACELERACIÓN**
- **COMBUSTIBLE**

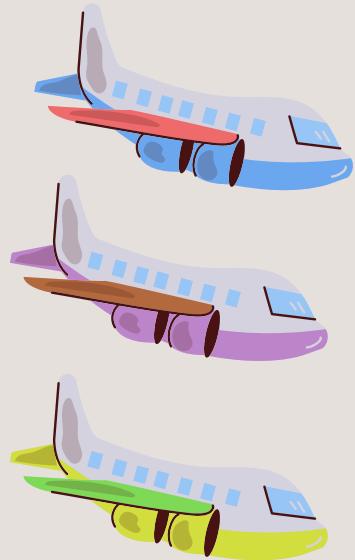
CONSTANTES

- **MARCA**
- **COLOR**
- **POTENCIA**
- **ASIENTOS**

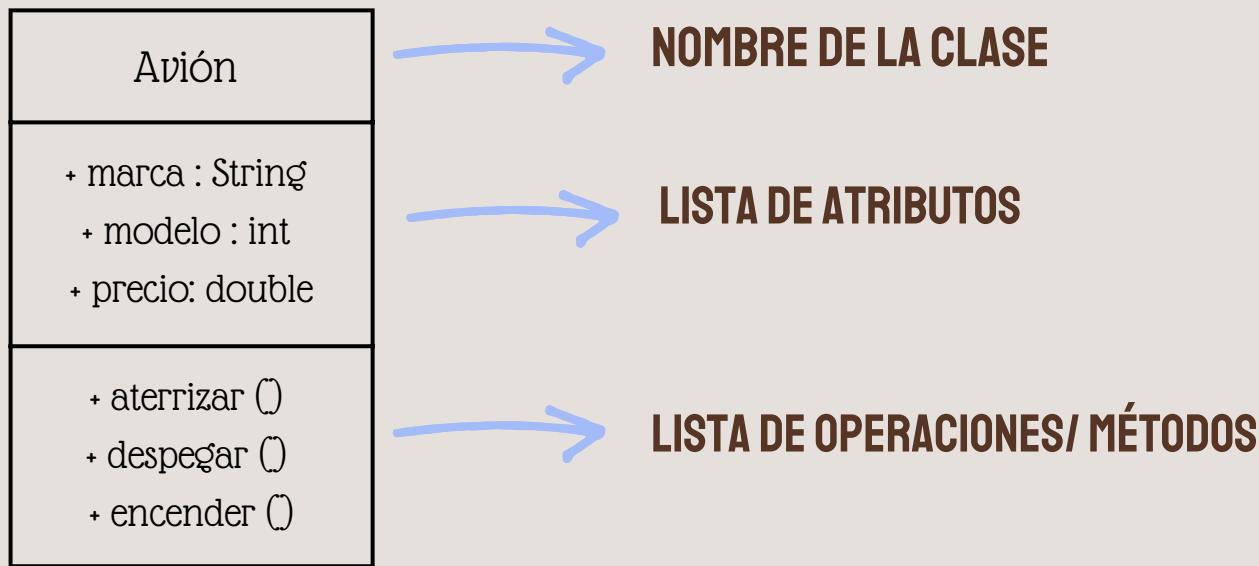


Además de representar las características de cada objeto, a través de una clase se puede definir un tipo de modelo para representar objetos del mismo tipo.

Como un avión pertenece a la clase de “aviones”, cualquier objeto está encontrado dentro de una clase. Podemos relacionar a las clases como fábricas de objetos que realizan el mismo producto, a través de esta idea, se abstrae el concepto del objeto para poder tener una representación del proceso de programación en el mundo real.



REPRESENTACIÓN DE UNA CLASE



INSTANCIA

Al definir una clase, se pueden crear objetos a partir de ella, a este proceso se le conoce como crear instancias de una clase o instanciar una clase. Por ejemplo al crear una fábrica de coches, pelotas o muebles, se empiezan a conseguir las piezas necesarias para crear estos objetos, se instancian las clases para obtener los productos.



Para ello, el sistema almacena suficiente memoria para el objeto y sus atributos

Una instancia se puede también definir como un objeto, un elemento que pertenece a una clase. Cada objeto o instancia tiene su propia copia de las variables definidas en la clase de la que son instanciados y comparten el mismo comportamiento o funciones de los métodos, pero cada objeto cuenta con valores diferentes para sus atributos y es independiente de los demás objetos de la clase.

En Java para crear una instancia se utiliza el operador “new” seguido del nombre de la clase y un par de paréntesis.

```
Avion avionBoeing = newAvion( );
```

Clase Objeto de la clase Avión Instanciación



MÉTODOS

Un método es un conjunto de instrucciones que realizan una tarea específica y determinan el comportamiento de los objetos de una clase, sirven para encapsular cierta acción que podamos llamar desde diferentes sitios sin necesidad de repetir el código.

Todos los métodos son funciones, lo que significa que regresan un dato. Sólo si definimos un método tipo “void”, indica que no devolverá nada.

Así como hay variables de instancia y de clase, también hay métodos de instancia y de clase. En los métodos de instancia, un objeto llama a un método para realizar una determinada tarea. En los de clase, el método se llama desde la propia clase.

Cada método tiene un nombre, y es este el que se usa para llamar al método, se puede usar casi cualquier nombre evitando las palabras clave de Java o métodos ya definidos del programa.

PARÁMETROS

Los parámetros son datos de entrada que pueden ser establecidos al llamar al método. Un método puede recibir ninguno, uno o más parámetros. Dentro del cuerpo del método, los parámetros son usados como variables.

```
int sumar (int A, int B)
```



DECLARACIÓN DEL MÉTODO

Un método está formado por un encabezado y un cuerpo. Para declarar el encabezado de un método, simplemente se escribe el tipo de dato que retorna, el nombre del método y entre paréntesis la lista de parámetros.

EJEMPLOS DE ENCABEZADOS DE MÉTODOS

```
void imprimir();
```

tipo
de dato

no tiene parámetros
nombre del método

este método no devuelve valores

```
int restar(int a, int b);
```

tipo
de dato

tiene 2 parámetros
nombre del método

este método recibe dos parámetros
de tipo int y devuelve un valor int

El cuerpo de un método indica una secuencia de instrucciones separadas por punto y comas. El cuerpo va a estar desarrollado dentro de una llave que abre y otra que cierra { }. La secuencia puede estar vacía {}.

EJEMPLOS DE CUERPOS DE MÉTODOS

```
void imprimir() {};
```



este método es válido pero está vacío y no lleva a cabo ninguna acción

```
int restar(int a, int b)
{
    return a-b
};
```



Este método recibe dos números y regresa la resta del primero menos el segundo

EJEMPLOS PARA LLAMAR MÉTODOS

```
// Clase con método sumar

public class Calculadora {
    public static int sumar(int a, int b)
    {
        return a + b;
    }
}

int suma = Calculadora.sumar(10, 5);
```



Se define el método sumar dentro de la clase Calculadora



Se llama el método sumar con los argumentos 10 y 5.

ARGUMENTOS

Cuando llamamos a un método que posee parámetros, debemos aportar valores para cada parámetro, a estos valores se les conoce como argumentos.

MÉTODOS ESTÁTICOS

Son métodos que pertenecen a la clase en sí misma, no a instancias individuales de la clase. Esto significa que puedes llamar a un método estático sin necesidad de crear un objeto de la clase.

```
public class Ejemplo {  
  
    public static void saludar() {  
        System.out.println("Hola desde un  
método estático");  
    }  
}  
  
// Llamar al método  
Ejemplo.saludar();
```

MÉTODOS MIEMBRO

Son métodos que pertenecen a una instancia (u objeto) de la clase. Para invocar un método miembro, necesitas crear una instancia de la clase.

```
public class Ejemplo {  
  
    public void saludar(  
        System.out.println("Hola desde  
un método miembro");  
    }  
}  
  
// Llamar al método  
Ejemplo objeto = new Ejemplo();  
objeto.saludar();
```

Aquí creamos una instancia de la clase, es decir un objeto perteneciente a la misma.

PARÁMETROS

Los parámetros son datos de entrada que pueden ser establecidos al llamar al método. Un método puede recibir ninguno, uno o más parámetros. Dentro del cuerpo del método, los parámetros son usados como variables.

```
int sumar (int A, int B)
```



DECLARACIÓN DEL MÉTODO

Un método está formado por un encabezado y un cuerpo. Para declarar el encabezado de un método, simplemente se escribe el tipo de dato que retorna, el nombre del método y entre paréntesis la lista de parámetros.

Todos los lenguajes orientados a objetos proporcionan mecanismos que ayudan a implementar el modelo orientado a objetos, los cuales son:

- **ENCAPSULACIÓN**

La encapsulación es el mecanismo que permite unir el código junto con los datos que manipula, y mantiene a ambos a salvo de las interferencias exteriores y de un uso indebido.

- **HERENCIA**

La herencia es el proceso por el cual un objeto adquiere las propiedades de otro objeto. Esto es importante, ya que supone la base del concepto de clasificación jerárquica.

- **POLIMORFISMO**

Permite que un objeto pueda tomar múltiples formas, proporcionando flexibilidad y extensibilidad al código. En términos simples, significa que un método del código puede comportarse de diferentes maneras dependiendo de las variables utilizadas.

PILARES DE LA PROGRAMACIÓN ORIENTADA A OBJETOS

Todos los lenguajes orientados a objetos proporcionan mecanismos que ayudan a implementar el modelo orientado a objetos, los cuales son:

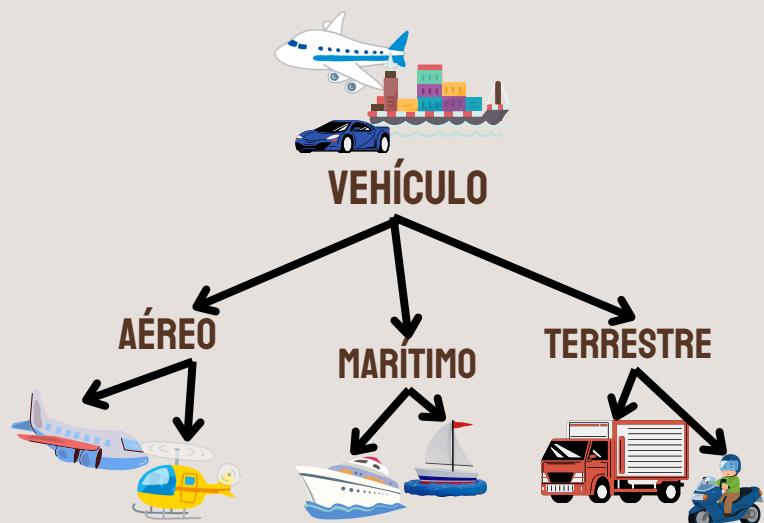
- **ENCAPSULACIÓN**

La encapsulación es el mecanismo que permite unir el código junto con los datos que manipula, y mantiene a ambos a salvo de las interferencias exteriores y de un uso indebido.



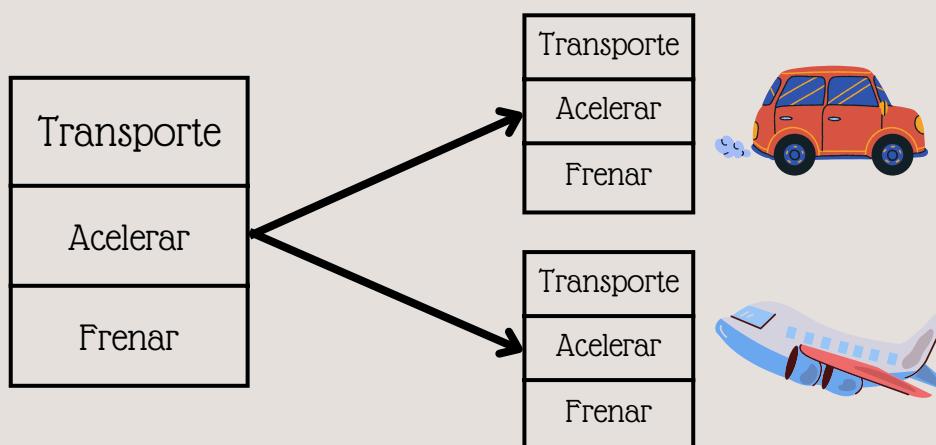
- **HERENCIA**

La herencia es el proceso por el cual un objeto adquiere las propiedades de otro objeto. Esto es importante, ya que supone la base del concepto de clasificación jerárquica.



- **POLIMORFISMO**

Permite que un objeto pueda tomar múltiples formas, proporcionando flexibilidad y extensibilidad al código. En términos simples, significa que un método del código puede comportarse de diferentes maneras dependiendo de las variables utilizadas.



ESTRUCTURA DE UN PROGRAMA

Un programa da instrucciones para que un ordenador ejecute ciertas acciones escritas por el programador. Los programas están compuestos por diferentes elementos.

• COMENTARIOS

El programa generalmente comienza con un comentario.

Los comentarios son ignorados por el compilador pero son útiles para los programadores, pues ayudan a explicar conceptos importantes de un programa y mejoran su comprensión. Se puede escribir todo lo que se desee, el texto puede abarcar una o más líneas.

/* inicio comentario */ fin comentario

• DEFINICIÓN DE CLASE

La primera línea después del primer comentario define una clase, la definición comienza por una llave { y termina con la otra }.

• DEFINICIÓN DE MÉTODO

Después de definir la clase se sigue con la definición del método “main () ”. Todos los programas incluyen el método main, el cual indica las sentencias a realizar cuando se ejecuta un programa.

Las sentencias de un método están delimitadas por una llave inicial { y una final }.

• SENTENCIA

Las sentencias son las instrucciones para ejecutar por el programa, todas deben terminar con el símbolo punto y coma, que indica al compilador que se ha finalizado la sentencia.

• IDENTIFICADORES

Existen reglas para elegir los nombres que se utilizan como identificadores de clases, de variables o de métodos.

- 1. Los nombres de las variables y los métodos comienzan en minúsculas. Si es un nombre compuesto cada palabra debe empezar con mayúscula y no debe usarse el guión bajo para separar las palabras.

**NOMBRES DE VARIABLES Y
MÉTODOS VÁLIDOS**

calcularArea	suma	getNombre
setNombre	calcularResta	perimetro

- 2. Los nombres de las clases empiezan siempre con mayúsculas. En los nombres compuestos cada palabra empieza con mayúscula y no se utiliza el guión bajo para separar las palabras.

**NOMBRES DE CLASE
VÁLIDOS**

PerimetroCuadrado	PresupuestoEmpresa
MiPrimerPrograma	Operaciones2
SueldoProfesor12	CuotaNueva

- 3. Los nombres de constantes se escriben en mayúsculas. En nombres compuestos se usa el guión bajo “_” para separar las palabras

**NOMBRES DE
CONSTANTES VÁLIDOS**

PI	MINIMO	MAXIMO_NUMERO
TOTAL_ELEMENTOS	DIAS_SEMANA	

EJEMPLO DE LA ESTRUCTURA DE UN PROGRAMA

```
public class HolaMundo {
    public static void main(String[] args) {
        // Llamada a un método estático
        saludar();

        // Llamada a un método con lógica simple
        int resultado = sumar(5, 7);
        System.out.println("La suma es: " + resultado);
    }
}
```

```

// Método estático para imprimir un saludo
public static void saludar() {
    System.out.println("¡Hola, mundo!");
}

// Método estático para sumar dos números
public static int sumar(int a, int b) {
    return a + b;
}
}

```

TIPOS DE VARIABLES

- **PRIMITIVOS**

Los tipos primitivos representan valores básicos

TIPO DE DATO	TAMAÑO
BYTE	8 BITS
SHORT	16 BITS
INT	32 BITS
LONG	64 BITS
FLOAT	32 BITS
DOUBLE	64 BITS
CHAR	16 BITS
BOOLEAN	1 BIT

Algunas de sus características:

- Tamaños fijos en bits, dependiendo del tipo.
- No pueden ser null, pero tienen valores por defecto si no se inicializan.
- Se usan para datos simples como números, caracteres y valores booleanos

26

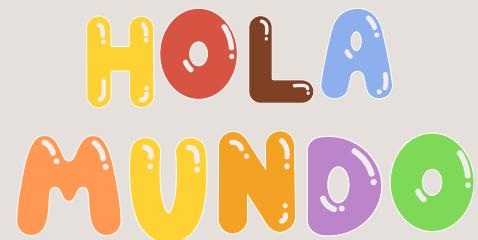
- **REFERENCIADOS**

Representan objetos o estructuras más complejas (como cadenas, arreglos, o clases definidas por el usuario).

TIPO DE DATO	TAMAÑO	DESCRIPCIÓN
STRING	VARIABLE	CADENA DE TEXTO COMO OBJETO (EJEMPLO: "HOLA")
ARRAYS	VARIABLE	ESTRUCTURA PARA ALMACENAR MÚLTIPLES VALORES DEL MISMO TIPO
CLASES	VARIABLE	INSTANCIAS DE OBJETOS DEFINIDOS POR EL USUARIO
INTERFACES	VARIABLE	REFERENCIAS A IMPLEMENTACIONES DE INTERFACES
ENUMERACIONES	VARIABLE	CONJUNTOS PREDEFINIDOS DE CONSTANTES

Algunas de sus características:

- Pueden ser nulos, lo que indica que no tienen una referencia válida.
- Se almacenan en el heap (memoria dinámica).



OPERADORES

• OPERADORES ARITMÉTICOS

OPERADOR	DESCRIPCIÓN	EJEMPLO	RESULTADO
+	SUMA	5 + 3	8
-	RESTA	5 - 3	2
*	MULTIPLICACIÓN	5 * 3	15
/	DIVISIÓN	10 / 2	5
%	MÓDULO (RESTO)	10 % 3	1

• OPERADORES DE COMPARACIÓN

OPERADOR	DESCRIPCIÓN	EJEMPLO	RESULTADO
==	IGUAL A	5 == 3	FALSE
!=	DIFERENTE DE	5 != 3	TRUE
>	MAYOR QUE	5 > 3	TRUE
<	MENOR QUE	5 < 3	FALSE
>=	MAYOR O IGUAL QUE	5 >= 5	TRUE
<=	MENOR O IGUAL QUE	5 <= 3	FALSE

• OPERADORES LÓGICOS

OPERADOR	DESCRIPCIÓN	EJEMPLO	RESULTADO
&&	AND LÓGICO (Y) REGRESA FALSE SI UNO DE LOS RESULTADOS ES FALSO.	(5 > 3) && (4 > 2)	TRUE
!	OR LÓGICO (O) DEVUELVE TRUE SI UNO DE LOS RESULTADOS ES VERDADERO.	6 < 3 3 = 3	TRUE



• OPERADORES DE ASIGNACIÓN

OPERADOR	DESCRIPCIÓN	EJEMPLO	EQUIVALENTE A
=	ASIGNACIÓN	A = 5	A = 5
+=	SUMA Y ASIGNACIÓN	A += 3	A = A + 3
-=	RESTA Y ASIGNACIÓN	A -= 3	A = A - 3
*=	MULTIPLICACIÓN Y ASIGNACIÓN	A *= 3	A = A * 3
/=	DIVISIÓN Y ASIGNACIÓN	A /= 3	A = A / 3
%=	MÓDULO Y ASIGNACIÓN	A %= 3	A = A % 3



• OPERADORES DE COMPARACIÓN

OPERADOR	DESCRIPCIÓN	EJEMPLO	RESULTADO
+	VALOR POSITIVO	+5	5
-	VALOR NEGATIVO	-5	-5
++	INCREMENTO	A++	A LUEGO A+1
--	DECREMENTO	--A	A-1 LUEGO A
!	NEGACIÓN LÓGICA	!(TRUE)	FALSE



ENTRADA Y SALIDA DE DATOS DESDE EL TECLADO

El flujo de entrada System.in, lee datos que son introducidos por el usuario a través del teclado. Mientras que el flujo de salida System.out muestra en el teclado los datos escritos en el programa.

Para indicar que se quiere leer y almacenar el valor de una variable, se pueden utilizar los métodos next() para leer una palabra, nextLine() para leer una línea y nextInt() para leer un número entero.

Para ocupar la clase Scanner en un programa se necesita importar la librería java.util.Scanner.

```
Scanner entrada1 = new Scanner (System.in);
```

declaración instancia

“entrada1” de la clase Scanner

flujo de entrada

nueva instancia de la clase

Scanner que permite el ingreso de datos

```
System.out.print("¿Cuál es el nombre del producto?");  
costoCompra= entrada1.nextFloat();  
  
System.out.print("¿Cuál es el costo de venta del  
producto?");  
costoVenta= entrada1.nextFloat();  
  
System.out.println("El producto " + "Coca-Cola" + ", el  
costo de compra es " + costoCompra + "y el costo de  
venta es" + costoVenta);
```

salida de texto que imprime en teclado para preguntar al usuario.

El producto se llama Coca-Cola, el costo de compra es 12.50 y el costo de venta es 15.00

como se vería en la interfaz del programa

salida de texto que imprime los datos introducidos por el usuario



FUNCIONES ACTUARIALES EN JAVA

Para los actuarios, Java puede ser una herramienta de gran ayuda en el desarrollo de modelos financieros, cálculos actuariales, y simulaciones estadísticas. A continuación, te comarto una lista de funciones y bibliotecas útiles que pueden ser utilizadas por actuarios en sus proyectos.

• INTERÉS COMPUESTO

Este código calcula el monto total después de aplicar interés compuesto durante varios períodos.

```
public class Finanzas {  
    public static double interesCompuesto(double principal,  
    double tasa, int periodos) {  
        return principal * Math.pow(1 + tasa, periodos);  
    }  
  
    public static void main(String[] args) {  
        double monto = interesCompuesto(1000, 0.05, 10); //  
        Usando como datos -> Principal: 1000, Tasa: 5%, Periodos:  
        10  
        System.out.println("Monto con interés compuesto:  
        " + monto);  
    }  
}
```

Explicación:

- Método `interesCompuesto`:
 - `principal`: El monto inicial de la inversión.
 - `tasa`: La tasa de interés por período (en este caso, por ejemplo, 0.05 para 5%).
 - `Periodos`: El número de períodos durante los cuales se aplica el interés.
 - `Math.pow (1 + tasa, periodos)`: Calcula el factor de interés compuesto, es decir $(1 + r)^n$



• GENERACIÓN DE NÚMEROS ALEATORIOS (DISTRIBUCIÓN NORMAL)

Este código genera números aleatorios que siguen una distribución normal con una media y desviación estándar específicas.

```
import java.util.Random;

public class Simulacion {

    public static double generarNormal(double media, double
desviacion) {

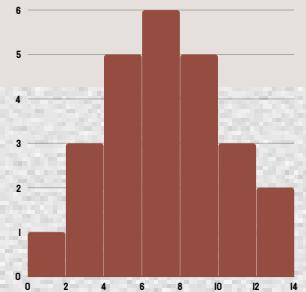
        Random random = new Random(); // Generador de
números aleatorios

        return media + desviacion * random.nextGaussian();
// Distribución normal
    }

    public static void main(String[] args) {

        for (int i = 0; i < 5; i++) {

            System.out.println(generarNormal(100, 15)); // Media: 100, Desviación estándar: 15
        }
    }
}
```



• Explicación:

- `Random random = new Random();`: Crea un generador de números aleatorios.
- `random.nextGaussian();`: Genera un número aleatorio que sigue una distribución normal estándar ($\text{media} = 0$, $\text{desviación} = 1$).
- Transformación de escala: $\text{media} + \text{desviacion} * \text{random.nextGaussian}();$
- Convierte la normal estándar a una distribución con la media y desviación dadas.

- **DISTRIBUCIÓN NORMAL ACUMULADA (USANDO APACHE COMMONS MATH)**

Este código calcula la probabilidad acumulada de una distribución normal estándar. Por ejemplo, esto es útil para calcular valores de tablas actuariales o evaluar percentiles.

```
import org.apache.commons.math3.distribution.NormalDistribution;

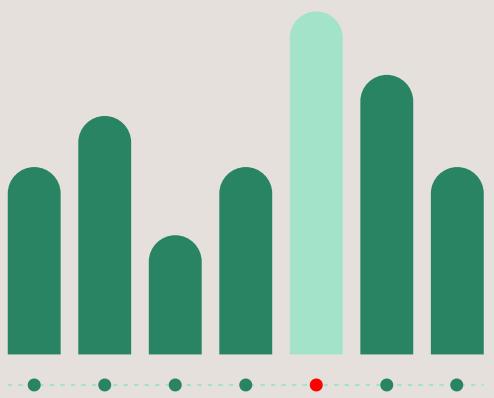
public class Estadistica {
    public static void main(String[] args) {
        NormalDistribution nd = new NormalDistribution(0, 1); // Media: 0, Desviación: 1
        double probabilidad = nd.cumulativeProbability(1.96); // P(Z <= 1.96)
        System.out.println("Probabilidad acumulada: " + probabilidad);
    }
}
```

- Explicación:
- NormalDistribution:
 - Clase de Apache Commons Math para trabajar con distribuciones normales.
 - new NormalDistribution(0, 1): Crea una distribución con media = 0 y desviación estándar = 1.
 - cumulativeProbability (1.96): Calcula $P(Z \leq 1.96)$ que representa el área bajo la curva hasta $Z=1.96$

RESULTADO ESPERADO:

$P(Z \leq 1.96) \approx 0.975$

(MUY CERCANO AL VALOR ESPERADO EN TABLAS ESTÁNDAR).



- **PROBABILIDAD DE SUPERVIVENCIA (TABLAS DE MORTALIDAD)**

Este código calcula la probabilidad de que una persona sobreviva hasta cierta edad, usando una tasa de mortalidad anual qx .

```
public class TablaMortalidad {  
    public static double probabilidadSupervivencia(int edad,  
double qx) {  
        return Math.pow(1 - qx, edad);  
    }  
  
    public static void main(String[] args) {  
        int edad = 30;  
        double qx = 0.02; // Probabilidad de mortalidad anual  
        System.out.println("Probabilidad de supervivencia: " +  
probabilidadSupervivencia(edad, qx));  
    }  
}
```

- Explicación:
- `probabilidadSupervivencia`:
 - Calcula $(1 - qx)^n$ donde:
 - qx : Probabilidad anual de fallecimiento.
 - n : Número de años (edad).
- Ejemplo:
- Si $qx= 0.02$, la probabilidad de sobrevivir 30 años es $(1 - 0.02)^{30} = 0.545$



- **SIMULACIÓN MONTE CARLO BÁSICA**

En las simulaciones actuariales, se puede usar Monte Carlo para estimar riesgos o evaluar distribuciones de resultados.

Por ejemplo el código para estimar la probabilidad de exceder un límite es el siguiente :

```

public class MonteCarlo {
    public static void main(String[] args) {
        int simulaciones = 10000;
        int exceden = 0;

        for (int i = 0; i < simulaciones; i++) {
            double resultado = generarNormal(100, 15); // Media: 100, Desviación: 15
            if (resultado > 120) {
                exceden++;
            }
        }

        double probabilidad = (double) exceden / simulaciones;
        System.out.println("Probabilidad de exceder 120: " + probabilidad);
    }

    public static double generarNormal(double media, double desviacion) {
        return media + desviacion * new java.util.Random().nextGaussian();
    }
}

```

- Explicación:

- Realizamos 10,000 simulaciones de una distribución normal. Contamos cuántas veces el resultado excede un límite (120 en este caso). Estimamos la probabilidad como:

$$\text{PROBABILIDAD} = \frac{\text{CASOS FAVORABLES}}{\text{SIMULACIONES TOTALES}}$$



RESULTADO ESPERADO :
UNA PROBABILIDAD CERCANA
AL 0.091 PARA MEDIA 100 Y
DESVIACIÓN 15.

CONCLUSIONES

Java es una herramienta complementaria en la automatización de cálculos matemáticos, permite desarrollar simulaciones avanzadas, así como obtener valores y probabilidades con facilidad.

Se puede trabajar en conjunto con bases de datos como SQL, para facilitar el manejo de datos actuariales, y a su vez con lenguajes estadísticos como R o Python para análisis más específicos.

BIBLIOGRAFÍA

- Unidad I: 1.2 Elementos del Lenguaje. (08 de 2014). Wordpress:<https://jmpovedar.files.wordpress.com/2014/08/introduccion-a-java.pdf>
- 1. INTRODUCCIÓN A JAVA. (s.f.).upv.es:<http://personales.upv.es/igil/java.PDF>
- Ladrón de Guevara, J. M. (s.f.). Fundamentos de programación en Java. tesuva.edu.co:https://www.tesuva.edu.co/phocadownloadpap/Fundamentos_de_programacion_en_Java.pdf
- Nakayama, A., & Solano Gálvez, J. A. (s.f.). Guía práctica de estudio 04: Clases y objetos. unam.mx: http://odin.fi.unam.mx/salac/practicasPOO/P_04-POO-Clases_y_Objetos.pdf
- Tania. (29 de 10 de 2015). Bases en que se fundamenta POO. toopropedeutico.blogspot.com:<http://toopropedeutico.blogspot.com/2015/10/bases-en-que-se-fundamenta-poo.html>
- Interpolados. (09 de 02 de 2017). EL MODELO ORIENTADO A OBJETOS. Wordpress: <https://interpolados.wordpress.com/tag/instancia-de-clase/> 47
- Walton, A. (06 de 05 de 2020). Métodos en Java con Ejemplos. javadesdecero.es:https://javadesdecero.es/poo/metodos-con-ejemplos/#1-%C2%BFQue_es_un_Metodo_en_Java
- Román, J. V. (s.f.). Programación Orientada a Objetos. uc3m.es: <http://www.it.uc3m.es/java/git-gisc/units/oo-herencia/slides/ProgramacionOrientadaAOBJETOS.pdf>