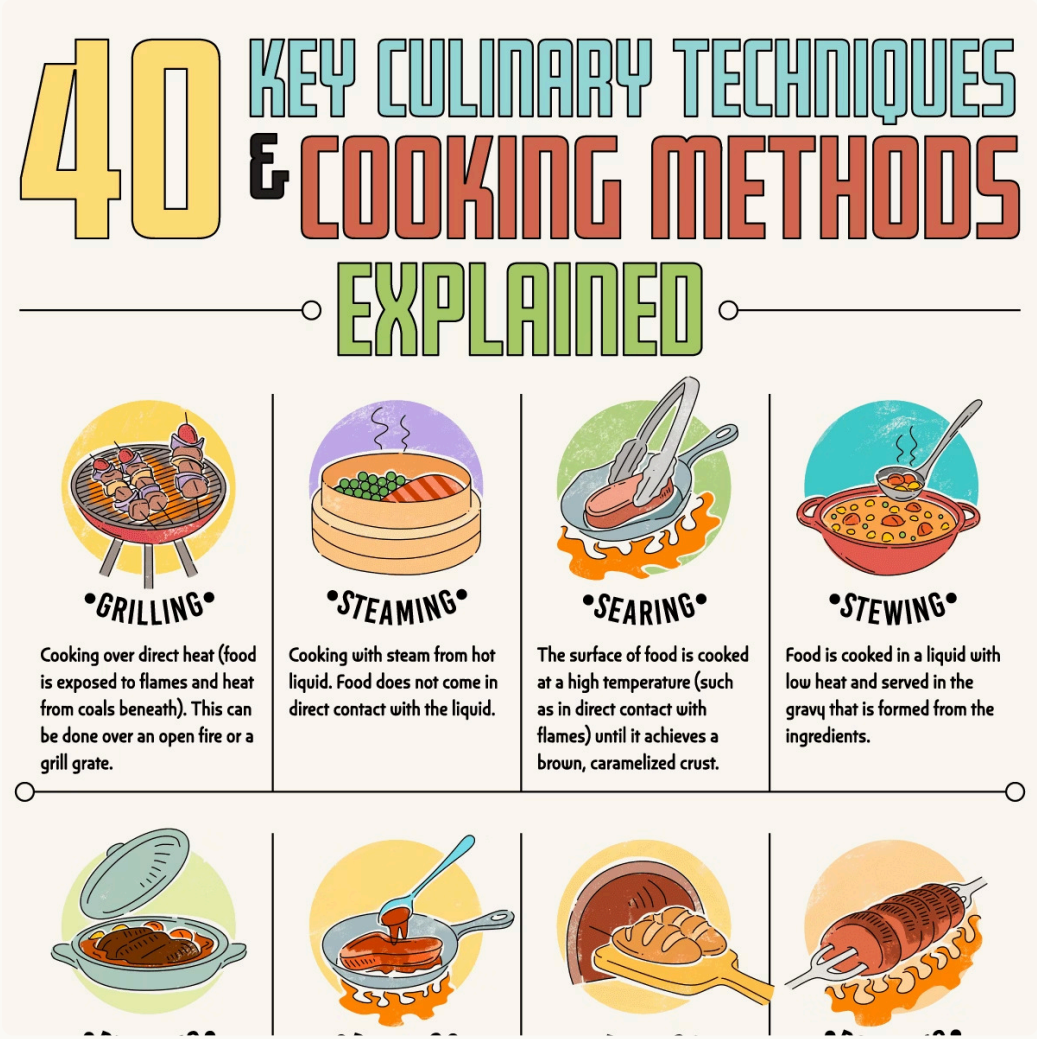
Fundamentos de la Programación Orientada a Objetos en Java

Una guía completa para estudiantes universitarios que desean dominar los conceptos esenciales de la POO y comenzar su viaje en el desarrollo de software profesional.

¿Qué es la Programación?

La programación es el arte de dar instrucciones precisas a una computadora para que realice tareas específicas. Es como ser un chef experimentado que tiene una receta detallada para crear el plato perfecto.

Cada línea de código representa una instrucción clara y específica, similar a los pasos de una receta culinaria que guían hacia el resultado deseado.



Programación Estructurada vs POO

Programación Estructurada

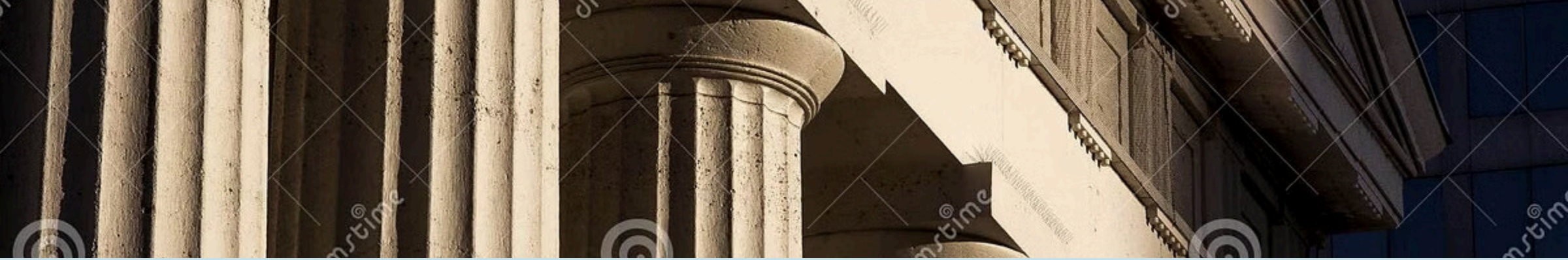
Divide el programa en funciones o subrutinas más pequeñas. Como separar una receta en secciones: preparar masa, hornear, decorar.

Útil para problemas sencillos, pero puede volverse difícil de manejar en proyectos grandes.

Programación Orientada a Objetos

Modela el mundo real usando "objetos" con atributos y comportamientos. Los ingredientes del pastel son objetos con propiedades y acciones.

Más flexible y reutilizable para proyectos complejos.



Los Cuatro Pilares de la POO

La Programación Orientada a Objetos se fundamenta en cuatro principios esenciales que permiten crear software robusto, mantenible y escalable. Estos pilares forman la base conceptual sobre la cual construiremos nuestras aplicaciones Java.

Abstracción

Simplificar la Realidad

La abstracción consiste en enfocarse en los detalles importantes e ignorar los irrelevantes. Es como concentrarse únicamente en los ingredientes esenciales del pastel e ignorar detalles como la marca específica de la harina.

Este principio nos permite manejar la complejidad del software identificando las características más relevantes de los objetos que modelamos.

Encapsulamiento

Protección de Datos

Ocultar los detalles internos de un objeto y exponer solo una interfaz pública. Como guardar la receta secreta en una caja fuerte y mostrar solo los pasos generales.

Control de Acceso

Permite controlar cómo se accede y modifica la información interna de nuestros objetos, garantizando la integridad de los datos.

Herencia

Pastel de Chocolate

Hereda base; añade cacao y ganache

Pastel

Atributos y métodos base compartidos

Pastel de Vainilla

Hereda base; añade esencia de vainilla

La herencia permite crear nuevas clases a partir de clases existentes, heredando atributos y métodos. Es como tener una receta base para pastel de vainilla y crear una receta para pastel de chocolate añadiendo ingredientes específicos.

Polimorfismo

Una Interfaz, Múltiples Formas

La capacidad de tratar objetos de diferentes clases de manera uniforme. Como poder usar el mismo método "hornear" para diferentes tipos de pasteles, aunque cada uno se hornee de manera ligeramente diferente.

Permite escribir código más flexible y extensible, facilitando el mantenimiento y la evolución del software.

Beneficios de la POO

Modularidad

Dividir un programa en módulos independientes (objetos) facilita la organización y el mantenimiento del código. Cada objeto es responsable de una funcionalidad específica.

Reutilización

Las clases y objetos se pueden reutilizar en diferentes partes del programa o en otros proyectos, ahorrando tiempo y esfuerzo de desarrollo.

Extensibilidad

Es fácil añadir nuevas funcionalidades creando nuevas clases o modificando las existentes sin afectar el resto del sistema.

Ventajas del Mantenimiento

La modularidad y el encapsulamiento de la POO facilitan significativamente la corrección de errores y la actualización del código. Los cambios en una clase no afectan necesariamente a otras partes del sistema.

Esta característica es especialmente valiosa en proyectos grandes donde múltiples desarrolladores trabajan simultáneamente en diferentes componentes del software.

Impacto en el Desarrollo Profesional

80%

Reducción de Errores

La encapsulación y modularidad reducen significativamente los errores de programación

60%

Tiempo de Desarrollo

La reutilización de código acelera el proceso de desarrollo de nuevas funcionalidades

90%

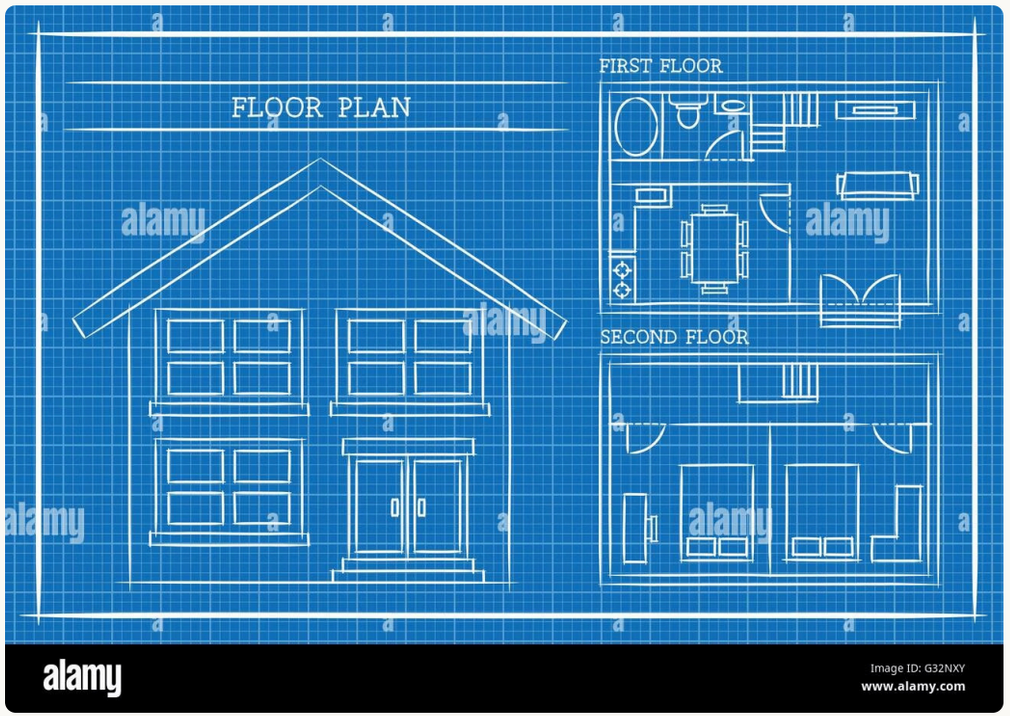
Mantenibilidad

El código POO es más fácil de mantener y actualizar a largo plazo

Clases: Los Planos de Nuestros Objetos

Definición de Clase

Una clase es como un plano arquitectónico que define la estructura y comportamiento que tendrán todos los objetos creados a partir de ella. Define los atributos (datos) y métodos (comportamientos) que caracterizarán a cada instancia.




```
public class Coche {
 String marca;
 String modelo;
 int año;

 public void acelerar() {
 System.out.println("¡Acelerando!");
 }
}
```

Anatomía de una Clase Java

1

Declaración de Clase

La palabra clave 'public class' seguida del nombre de la clase define el inicio de nuestra plantilla

2

Atributos

Variables que almacenan el estado y las características de cada objeto de la clase

3

Métodos

Funciones que definen las acciones y comportamientos que pueden realizar los objetos

Objetos: Instancias Reales

Un objeto es una instancia concreta de una clase. Si la clase es el plano del coche, el objeto es el coche físico que construimos usando ese plano. Cada objeto tiene sus propios valores para los atributos definidos en la clase.

```
Coche miCoche = new Coche();
miCoche.marca = "Toyota";
miCoche.modelo = "Corolla";
miCoche.acelerar(); // Ejecuta el método
```

Estado e Identidad de Objetos

Estado

El estado de un objeto se define por los valores actuales de todos sus atributos en un momento específico. El estado puede cambiar durante la ejecución del programa.

Identidad

Cada objeto tiene una identidad única que lo distingue de otros objetos, incluso si tienen exactamente el mismo estado. Son entidades independientes en memoria.

Creando Múltiples Objetos

```
public class Main {
    public static void main(String[] args) {
        Coche miCoche = new Coche();
        miCoche.marca = "Toyota";

        Coche otroCoche = new Coche();
        otroCoche.marca = "Ford";

        // Dos objetos independientes
        miCoche.acelerar();
        otroCoche.acelerar();
    }
}
```

Cada objeto mantiene su propio estado independiente. Modificar los atributos de un objeto no afecta a otros objetos de la misma clase.

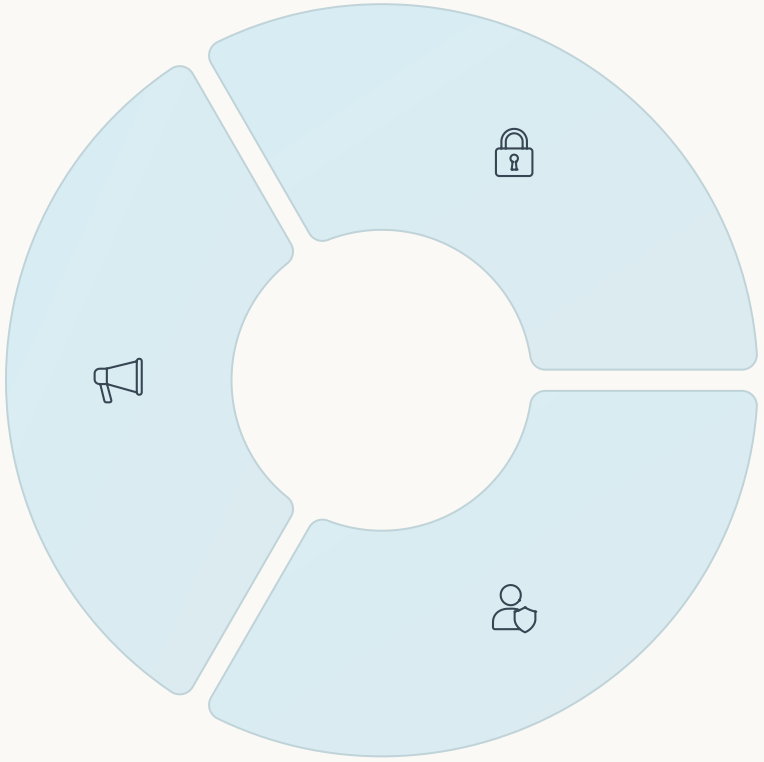
Encapsulamiento y Modificadores de Acceso

El encapsulamiento es uno de los principios fundamentales que nos permite controlar el acceso a los datos y métodos de nuestras clases, protegiendo la integridad de la información y facilitando el mantenimiento del código.

Modificadores de Acceso en Java

public

Accesible desde cualquier parte del programa. Máxima visibilidad.



private

Accesible solo desde dentro de la misma clase. Máxima protección.

protected

Accesible desde la misma clase, subclases y el mismo paquete.

Ejemplo Práctico: Cuenta Bancaria

```
public class CuentaBancaria {
    private String titular;
    private double saldo;

    public void depositar(double cantidad) {
        saldo += cantidad;
    }

    public double obtenerSaldo() {
        return saldo;
    }
}
```

Los atributos privados solo pueden ser modificados a través de métodos públicos controlados, garantizando que las operaciones sean válidas y seguras.

Beneficios del Encapsulamiento

- Protege los datos de modificaciones accidentales o maliciosas
- Permite validar los datos antes de almacenarlos
- Facilita el mantenimiento al centralizar el control de acceso
- Reduce el acoplamiento entre diferentes partes del código
- Mejora la legibilidad y comprensión del código

Métodos: El Comportamiento de los Objetos

01

Definición

Los métodos definen las acciones que pueden realizar los objetos. Pueden recibir parámetros y devolver valores.

02

Parámetros

Datos de entrada que el método necesita para realizar su función específica.

03

Valor de Retorno

Resultado que el método devuelve después de completar su procesamiento.

Ejemplo: Clase Rectángulo

```
public class Rectangulo {
 private double ancho;
 private double alto;

 public Rectangulo(double ancho, double alto) {
 this.ancho = ancho;
 this.alto = alto;
 }

 public double calcularArea() {
 return ancho * alto;
 }

 public double calcularPerimetro() {
 return 2 * (ancho + alto);
 }
}
```

Getters y Setters

Métodos Getter

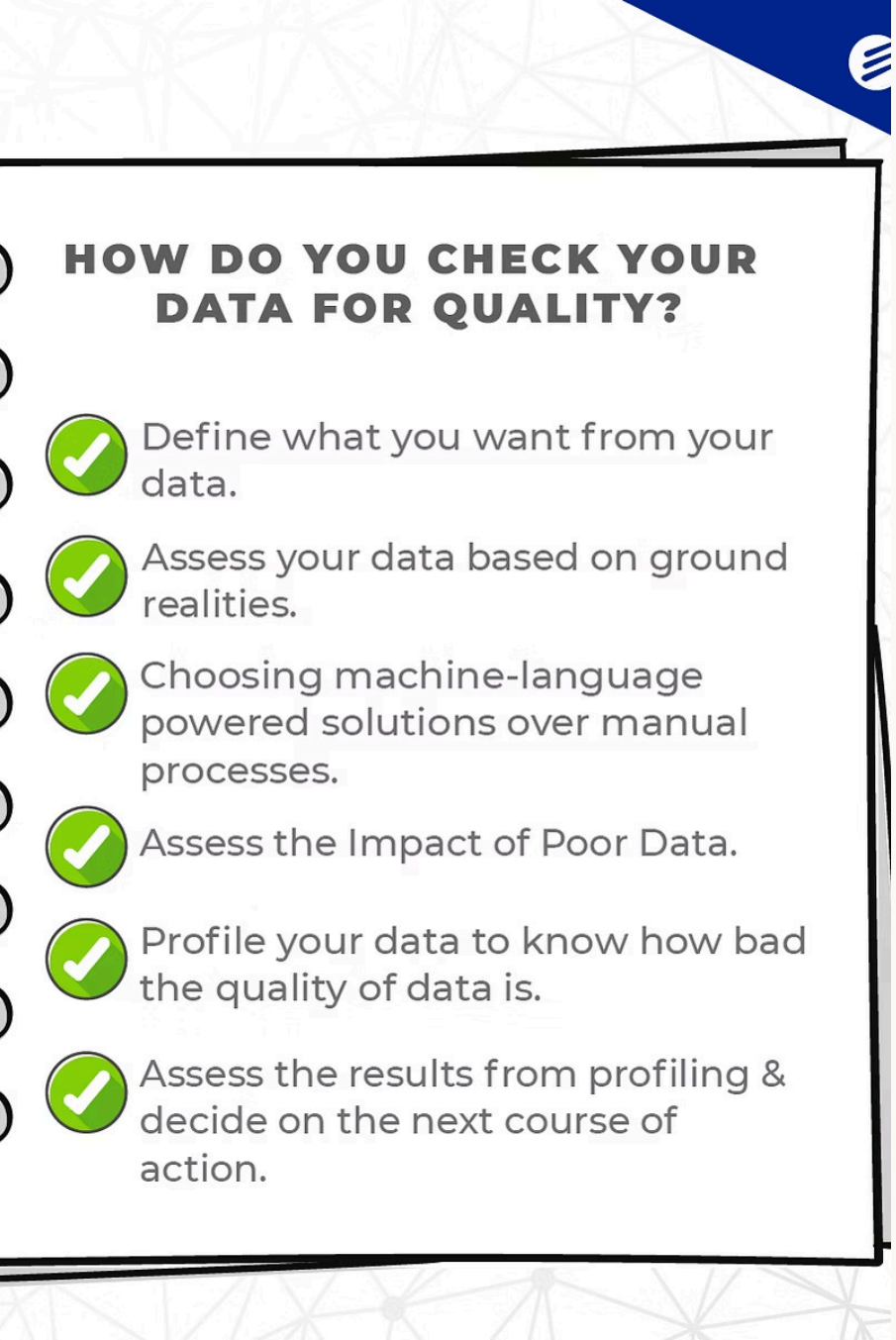
Permiten acceder al valor de atributos privados de forma controlada. Siguen la convención getNombreAtributo().

```
public String getNombre() {
    return nombre;
}
```

Métodos Setter

Permiten modificar el valor de atributos privados con validaciones. Siguen la convención setNombreAtributo().

```
public void setEdad(int edad) {
 if (edad >= 0) {
 this.edad = edad;
 }
}
```



Validación en Setters

Los métodos setter nos permiten implementar validaciones antes de modificar los atributos, garantizando que los datos almacenados cumplan con las reglas de negocio establecidas.

```
public void setEdad(int edad) {
 if (edad >= 0 && edad <= 120) {
 this.edad = edad;
 } else {
 System.out.println("Edad inválida");
 }
}
```

Ejemplo Integral: Sistema de Estudiantes

Modelado del Mundo Real

Vamos a crear un sistema que modele estudiantes y cursos, demostrando cómo la POO nos permite representar entidades del mundo real de forma natural e intuitiva.


```
public class Estudiante {
 private String nombre;
 private int id;
 private String carrera;

 public void estudiar() {
 System.out.println(nombre + " está estudiando.");
 }
}
```

Sistema de Reservas de Vuelos

```
public class Vuelo {
    private String numeroVuelo;
    private String origen;
    private String destino;
    private int asientosDisponibles;

    public boolean reservarAsiento() {
        if (asientosDisponibles > 0) {
            asientosDisponibles--;
            return true;
        }
        return false;
    }
}
```

Este ejemplo muestra cómo encapsular la lógica de negocio dentro de los métodos, manteniendo la consistencia de los datos.

Tienda de Libros Digital

```
public class Libro {
    private String titulo;
    private String autor;
    private double precio;
    private int stock;

    public boolean venderLibro() {
        if (stock > 0) {
            stock--;
            return true;
        }
        return false;
    }
}
```

Preguntas para Reflexionar

> ¿Cómo se relaciona la POO con la forma en que percibimos y organizamos el mundo real que nos rodea?

> ¿Por qué es fundamental el encapsulamiento y cómo contribuye directamente a la seguridad y mantenibilidad del código?

Más Reflexiones Importantes

¿Cómo la herencia y el polimorfismo fomentan la reutilización eficiente del código en proyectos complejos?

¿Qué estrategias específicas puedes desarrollar para identificar clases y objetos relevantes al analizar un problema de programación?

¡Enhorabuena!

Has completado el recorrido por los fundamentos de la Programación Orientada a Objetos en Java. Ahora posees las bases sólidas necesarias para crear software robusto, mantenible y escalable.

¡El viaje hacia la maestría en programación Java acaba de comenzar! 🚀