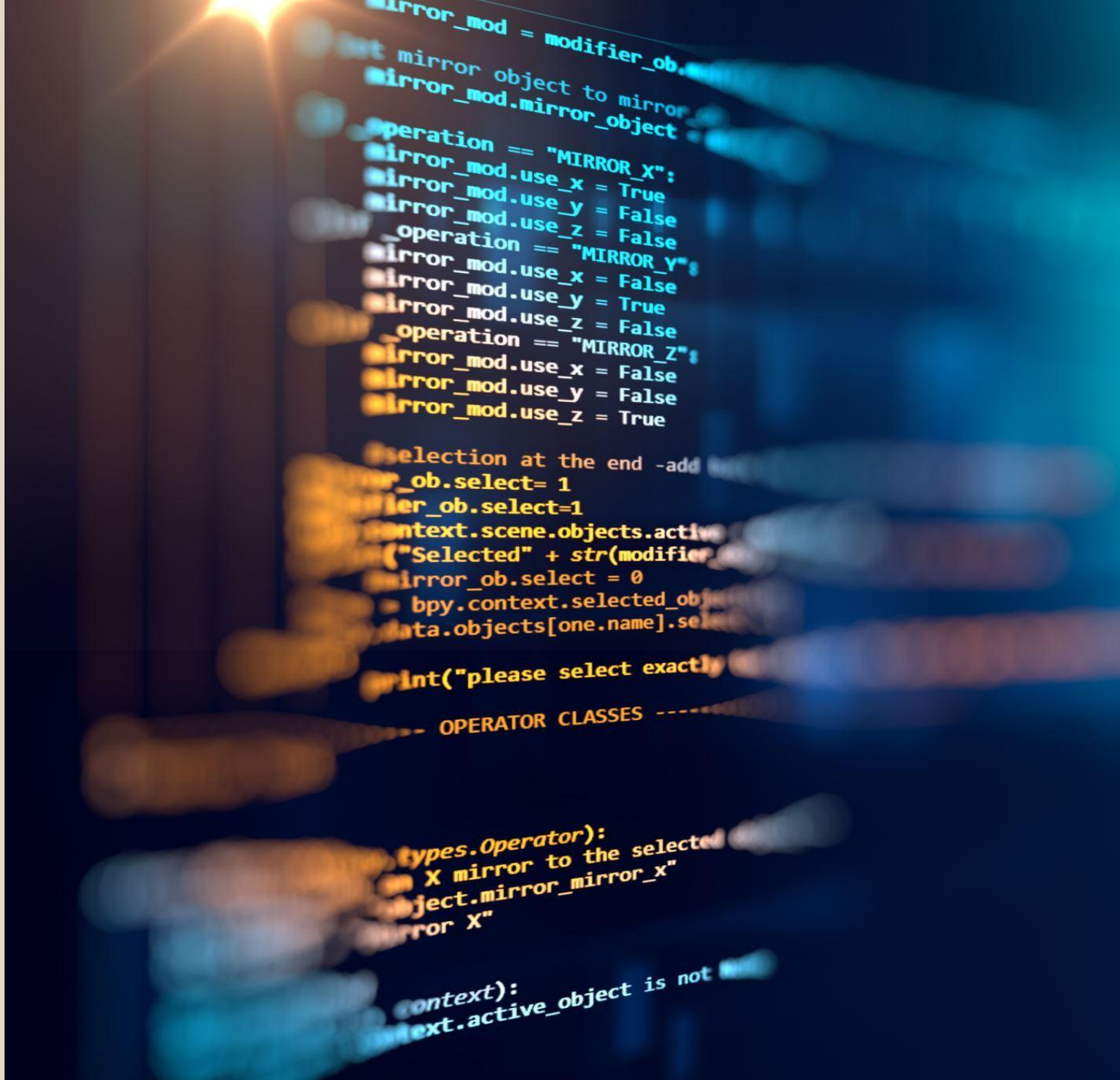
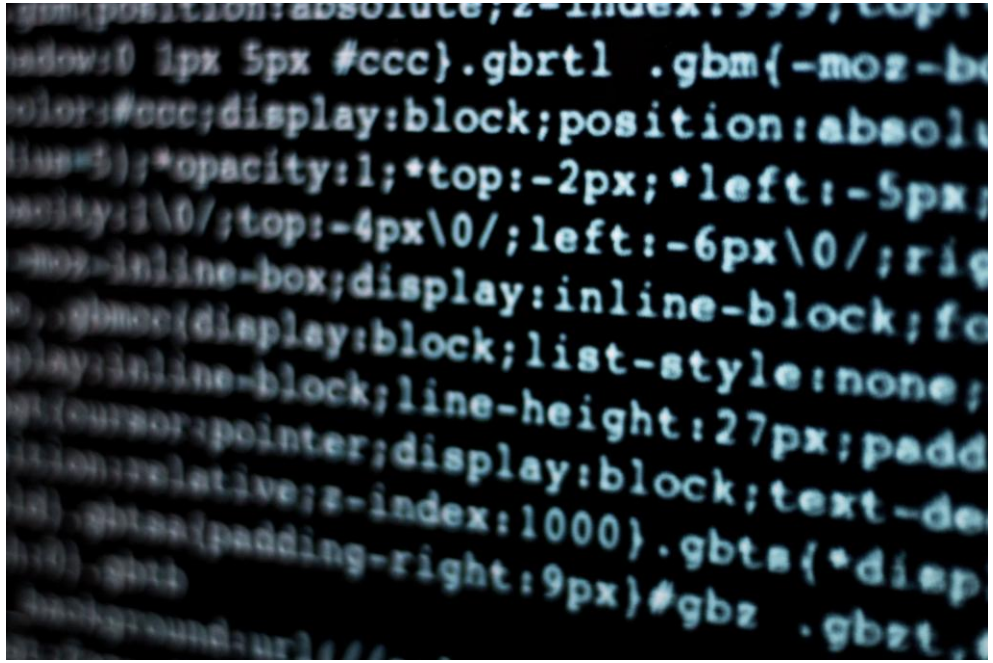
FUNDAMENTOS DEL DESARROLLO WEB EN ENTORNO CLIENTE: TYPESCRIPT Y PROGRAMACIÓN ORIENTADA A OBJETOS

Conceptos clave para crear aplicaciones web modernas y eficientes



INTRODUCCIÓN A TYPESCRIPT

¿QUÉ ES TYPESCRIPT Y SUS VENTAJAS FRENTE A JAVASCRIPT?



Superconjunto de JavaScript

TypeScript amplía JavaScript con características adicionales para mejorar la seguridad y mantenibilidad del código.

Tipado estático y detección de errores

Permite definir tipos de datos y detectar errores antes de la ejecución, evitando fallos en tiempo real.

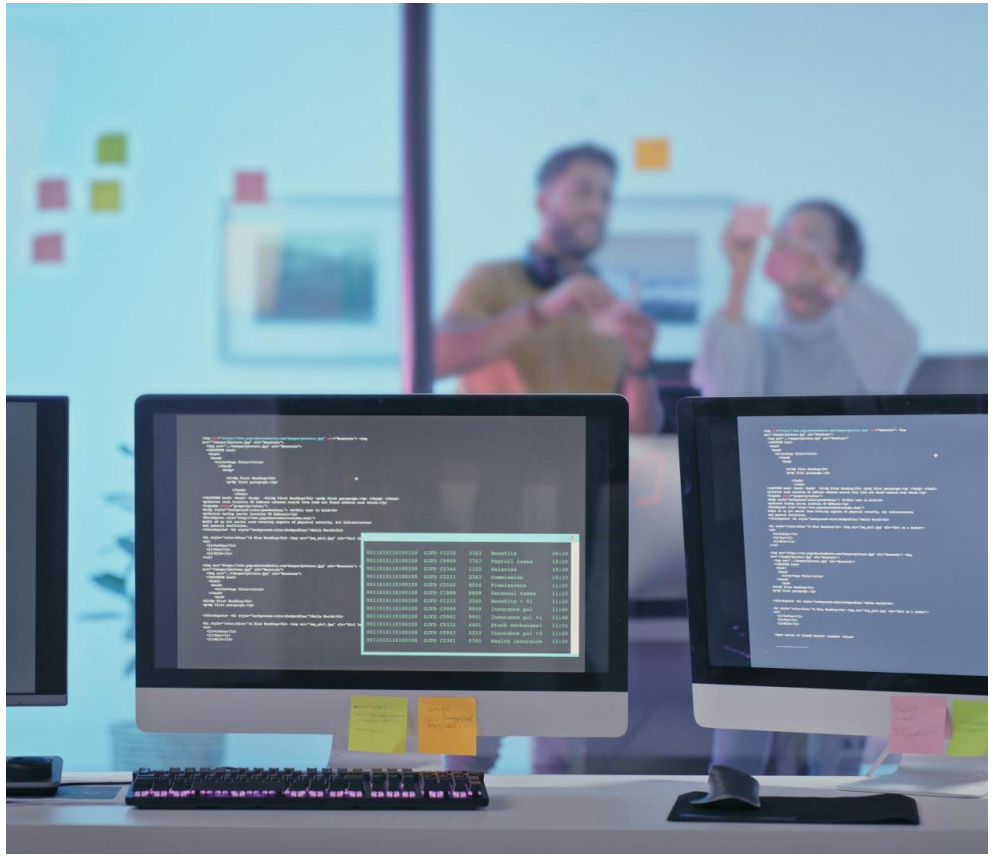
Mejora en la mantenibilidad

Uso de interfaces, clases y tipado para mejorar la estructura y claridad del código en proyectos colaborativos.

Compatibilidad y programación orientada a objetos

Compatible con JavaScript, permite usar librerías existentes y aplicar herencia y modificadores de acceso.

INSTALACIÓN, CONFIGURACIÓN Y USO CON FRAMEWORKS



Instalación y compilación básica

TypeScript se instala global o localmente, y sus archivos .ts se compilan a .js con el comando tsc.

Configuración con tsconfig.json

El archivo tsconfig.json configura opciones clave como versión de JavaScript, módulos y comprobación estricta.

Estructura del proyecto

La estructura típica incluye carpetas src para código fuente y dist para archivos generados por TypeScript.

Integración con frameworks

TypeScript se integra con Node.js, Angular, React y Vue.js, mejorando la calidad del código y robustez.

TIPOS BÁSICOS, ESTRUCTURAS DE DATOS Y FUNCIONES EN TYPESCRIPT

TIPOS PRIMITIVOS Y ESPECIALES

Tipos primitivos básicos

String, number y boolean son tipos primitivos que representan texto, números y valores lógicos respectivamente y son inmutables.

Tipos especiales null y undefined

Null y undefined indican ausencia de valor y su manejo puede configurarse con strictNullChecks para mayor seguridad.

Tipos any y unknown

Any desactiva el chequeo de tipos, mientras unknown acepta cualquier valor pero requiere verificación de tipo antes de usar.

Tipos void y never

Void es para funciones sin retorno y never para funciones que nunca terminan o lanzan errores, mejorando la predictibilidad.

ESTRUCTURAS DE DATOS Y FUNCIONES

Estructuras de datos en TypeScript

TypeScript soporta arrays y tuplas para almacenar colecciones con tipos precisos y ordenados.

Enumeraciones para constantes

Las enumeraciones permiten definir conjuntos de constantes con nombre, mejorando la legibilidad del código.

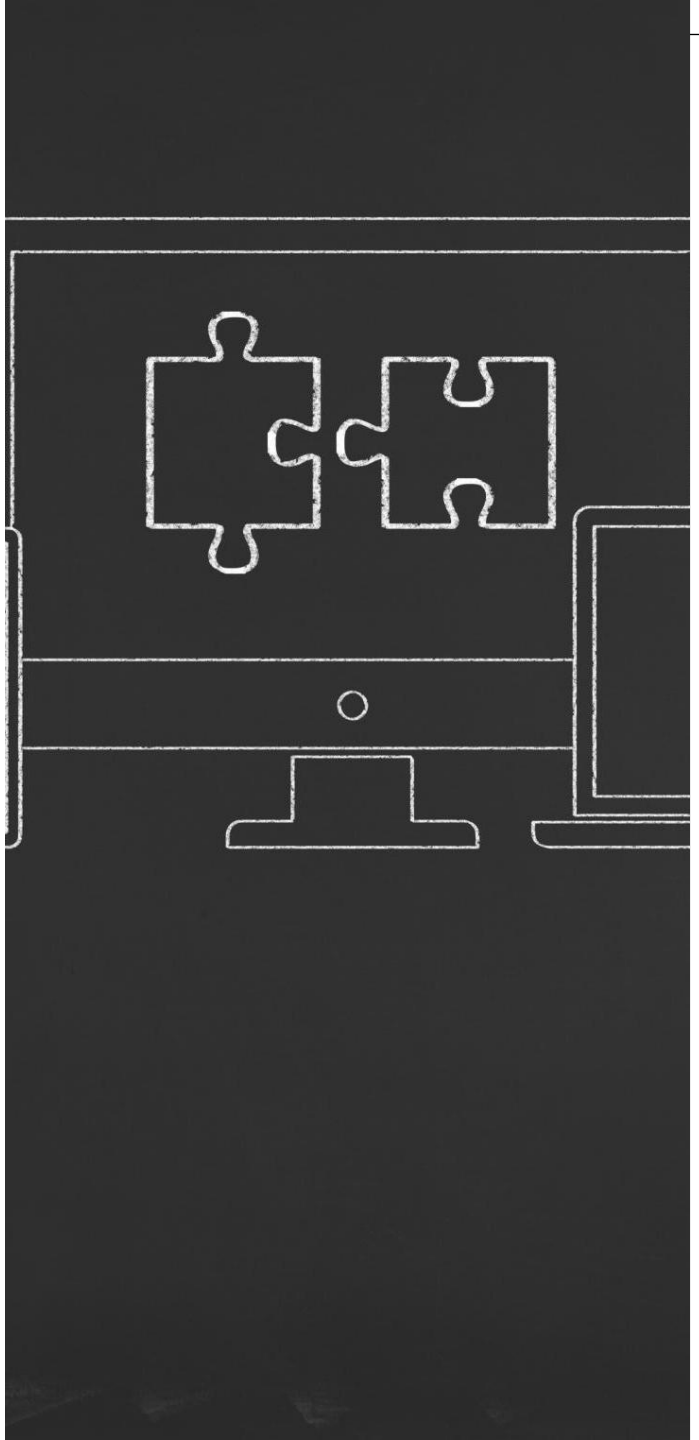
Funciones tipadas y seguras

Funciones con parámetros y retornos tipados facilitan la validación y reducen errores en desarrollo.

Tipos literales, uniones e inferencia

Tipos literales y uniones restringen valores; la inferencia deduce tipos automáticamente.

PROGRAMACIÓN ORIENTADA A OBJETOS



CLASES, CONSTRUCTORES Y MODIFICADORES DE ACCESO

Definición de Clase

Una clase es una plantilla que define atributos y métodos para crear objetos con características específicas.

Constructores en Clases

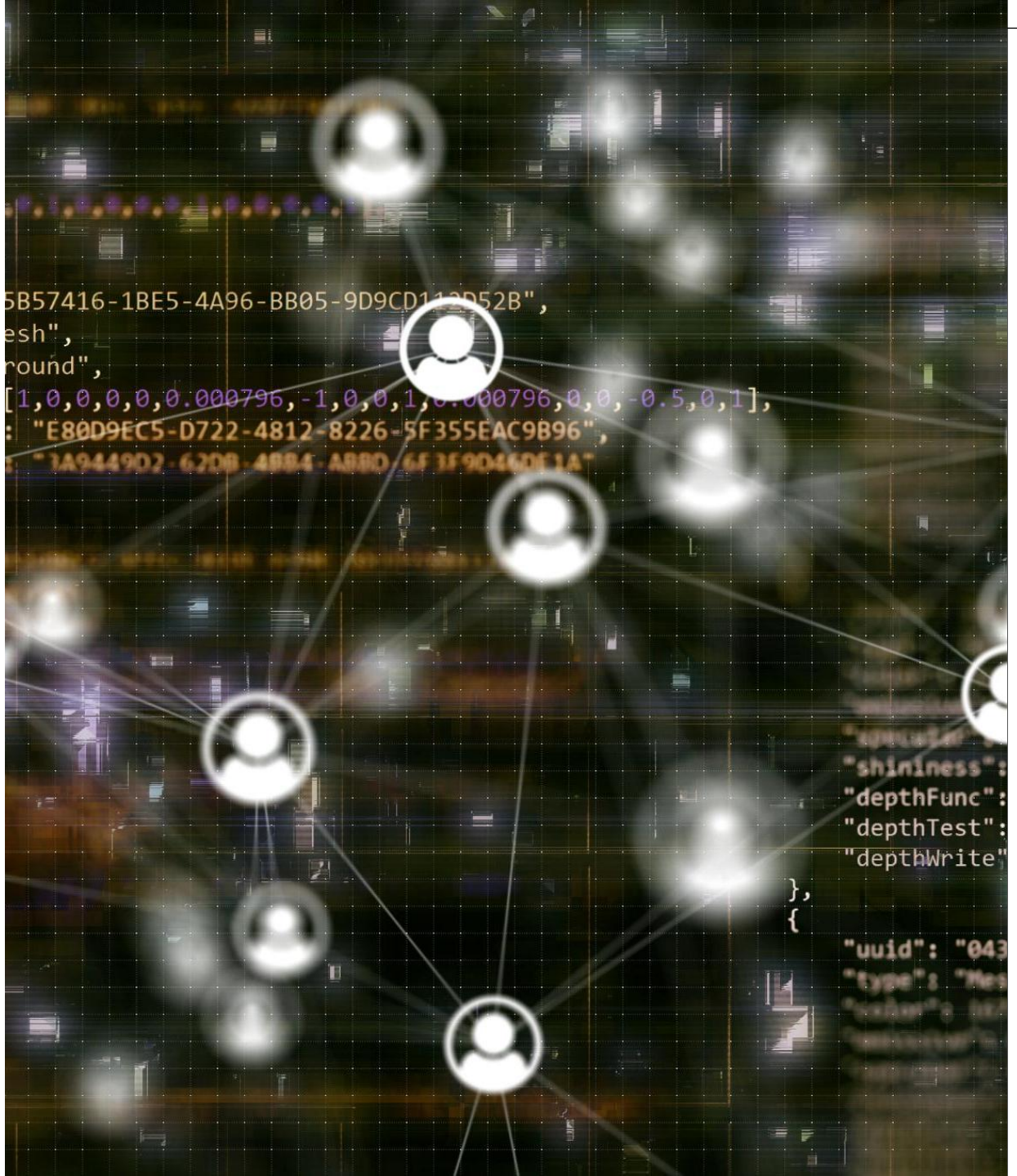
Los constructores inicializan objetos y permiten establecer valores iniciales, pudiendo ser sobrecargados.

Modificadores de Acceso

Public, private y protected controlan la visibilidad de atributos y métodos para fomentar la encapsulación.

Organización y Seguridad

Estas características facilitan código claro, seguro, reutilizable y fácil de mantener.



HERENCIA, POLIMORFISMO E INTERFACES

Concepto de Herencia

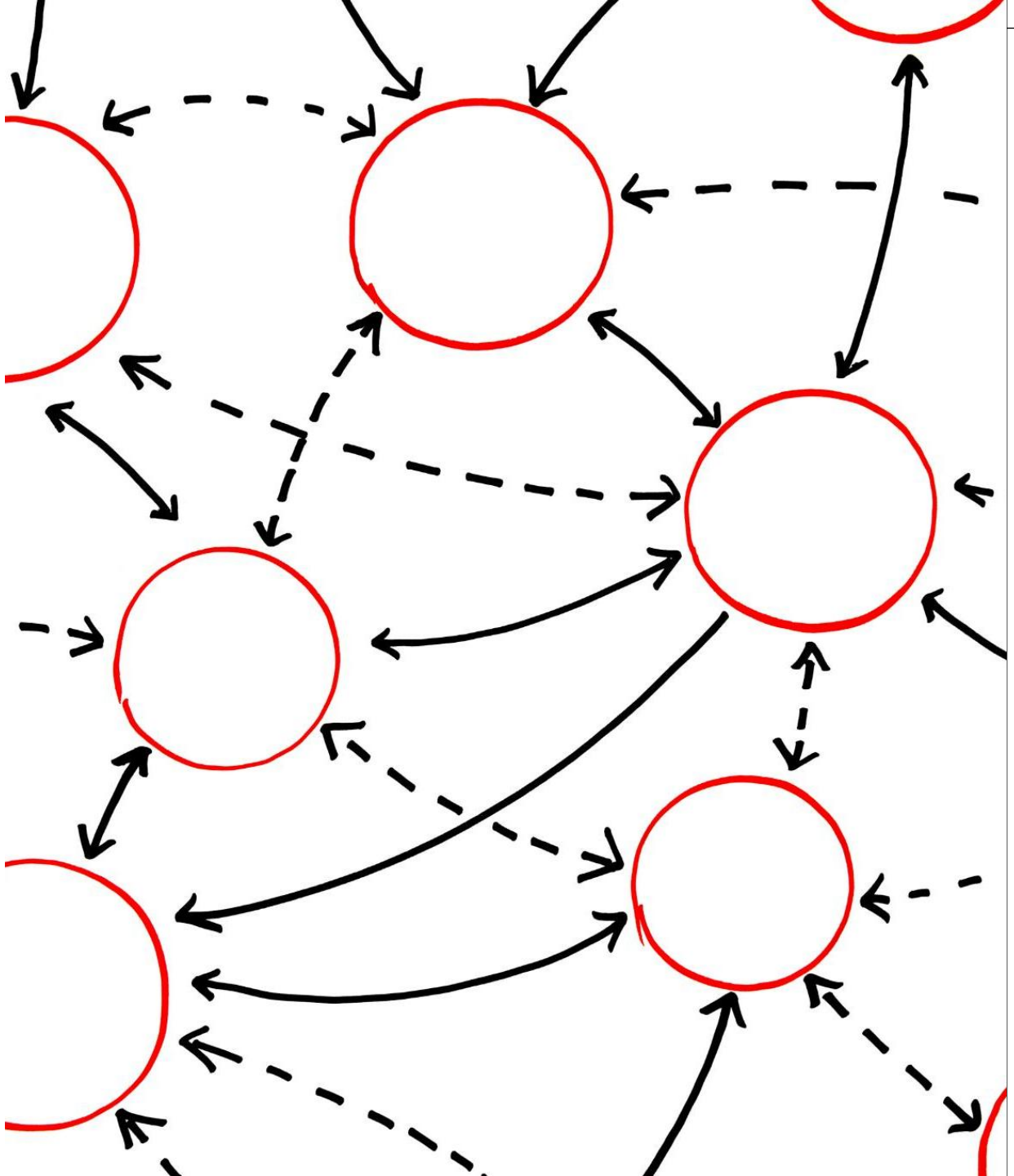
La herencia permite reutilizar atributos y métodos de una clase padre en una clase hija, creando jerarquías claras y eficientes.

Polimorfismo en programación

El polimorfismo permite que un método se comporte diferente según el objeto que lo utilice, con sobrecarga y sobrescritura.

Funciones de las Interfaces

Las interfaces definen contratos que obligan a implementar métodos, mejorando la flexibilidad y el desacoplamiento del código.



PROPIEDADES Y MÉTODOS ESTÁTICOS

Miembros estáticos de clase

Los miembros estáticos pertenecen a la clase y se comparten entre todas sus instancias sin necesidad de crear objetos.

Usos comunes

Se usan para definir constantes, contadores globales y funciones auxiliares accesibles sin instanciar la clase.

Limitaciones de métodos estáticos

Los métodos estáticos no pueden acceder directamente a atributos no estáticos, facilitando la organización y optimización del código.