Modulo 1 - Introducción

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

1- Que es POO? Paradigma que hace uso de los objetos para la construcción de los software. Es un estilo de programar, basicamente se busca plasmar la realidad con el codigo.

Como pensar en objetos? Enfocandonos en algo de la realidad, detallando sus atributos y sus comportamientos (acciones).

2- Paradigmas de la programación: Paradigma = modelo, ejemplo, patrón, es una forma o estilo de programar.

Tipos de paradigmas: Imperativa, declarativa, estructurada, orientada a objetos y funcional.

Imperativa= Instrucciones a seguir para dar solucion a un problema ej una receta

Declarativa = enfoque en el problema

Estructurada= La solucion a un problema sigue una secuencia de inicio a fin

Funcional = Divide el problema en diversas soluciones que serán ejecutadas por las funciones declaradas.

POO = construye soluciones basadas en objetos.

3- Clases y objetos

\* Una clase es una plantilla y un objetos es la instancia de la clase.

4- Principios POO

-SOn una serie de conceptos y fundamentos a tener en cuenta. Abstraccion, Encapsulamiento, herencia y polimorfismo

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Modulo 2 - Abstraccion

5- Es la forma de plasmar algo hacia el código para enfocarnos en su uso

6- Propiedades, métodos y el constructor. Los ejemplos que vamos a ver seran con el lenguaje de TS

class Smartphone {

color : string; --> Atributos

marca : string;

contructor(color: string, marca: string){ --> metodo que se ejcuta siempre cuando se inicializa la clase

this.color = color;

this.marca = marca;

}

llamar() : void {} --> metodos

}

7-

Propiedades

Representan al estado de nuestro objeto, se especifican de un tipo que podría ser boolean, numérico, string, etc.

Métodos

Es donde se definen los métodos de nuestra clase que realizan una cierta lógica.

Los métodos pueden recibir parámetros de entrada.

Los métodos si retornan un resultado será asociado a un tipo (boolean, number, string, etc) del caso contrario se definen como void (que no te retorna nada).

Constructor

Es el método que se ejecuta siempre al instanciar una clase

8- Instancia de una clase

let obj = new Smartphone('White', 'Iphone')

3 caracteristicas importante a tener en cuenta

Estado: propiedades de la clase

Comportamiento: Metodos dentro de la clase

Identidad: Cada instancia del objeto es unica

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Modulo 3 - Encapsulamiento

informacion de nueestro objeto (propiedades) que debe ser irrelevante para el usuario y no accesible para la modificación

Modificadores de acceso

En muchos lenguajes de programación orientada a objetos se conoce el concepto de modificadores de acceso, el cual puede modificar el acceso a una propiedad o método.

Públicos: son accesibles luego de instanciar la clase.

Privados: son accesibles dentro de la clase.

Protected: son accesibles a través de la herencia.

Getters & Setters

Para modificar directamente nuestras propiedades lo que se busca es que se haga a través de los métodos Getters y Setters, el cual varía la escritura según el lenguaje pero el concepto es el mismo.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Modulo 4 - Herencia

Es la cualidad de heredar caracteristicas de otra clase base para que no se repita las mismas propiedades o metodos.

EJ Clase BASE: Celular

Creo mi otra clase Android y hereda de Celular

Creo mi otra clase Iphone y hereda de Celular

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Modulo 5 - Polimorfismo

Las multiples formas que puede obtener un objeto si comparte la misma clase o interfaz

INTERFAZ!!! = Son un contrato que obliga a una clase a implementar las propiedades o métodos definidos, no tienen limites de implementacion. Las interfaces no tienen constructor, no implementan logica.

Clases abstractas = son clases que no se pueden instanciar, solo pueden ser implemetadas a través de la herencia.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Modulo 6 - Relaciones

Asociacion de uno a uno : Una clase mantiene una asociacion de uno a uno

de uno a muchos: Una clase mantiene una asociación con otra clase a traves de una coleccion

de muchos a mucho: La sociacion se da en ambos lados a travez de una coleccion

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Modulo 7 - PRINCIPIOS SOLID

Son recomendaciones para escribir codigo mantenible, escalable y robusto. Con los principios ganamos Alta cohesion (colaboracion entre clases) y bajo acomplamiento (Evitar que una clase dependa fuertemente de otra clase)

Pricipio de Responsabilidad unica : Una clase debe tener una razon para existir

Principio de abierto-Cerrado: Las piezas de software (Clase, funcion) deben estar abiertas para la extensión pero cerradas para la modificación.

Principio de sustitucion de Liskov: Las clases subtipos deberian ser reemplazables por sus clases padres. (herencia)

Principio de segregacion de interfaz: Varias interfaces funcionanmejor que una sola.

Principio de inversion de dependencia: Clases de alto nivel no deben depender de las clases bajo nivel (inyeccion de dependencia)