Clases en TypeScript

Las clases son las entidades fundamentales que se utilizan para crear componentes reutilizables. Las funcionalidades se transmiten a las clases y los objetos se crean a partir de clases.

Clases - Moldes para crear objetos con propiedades y métodos compartidos.

Propiedades - Características que definen al objeto.

```typescript
class Persona {
  nombre: string;
  edad: number;

  constructor(nombre: string, edad: number) {
    this.nombre = nombre;
    this.edad = edad;
  }

  saludar() {
    console.log(`Hola, me llamo ${this.nombre} y tengo ${this.edad} años.`);
  }
}

const Juan = new Persona("Juanita", 23);

Juan.saludar();
```

Métodos - Funciones que pertenecen a la clase/objeto.

Objetos - Instancias de una clase. Tienen propiedades y métodos propios.

Abstracción - Simplificar y mostrar solo los detalles importantes.

Crear una instancia de objeto:

```typescript
const Juan = new Persona("Juanita", 23);
```

Acceder a propiedades y métodos:

```typescript
juan.edad = 24;
juan.nombre = "Juanita"
juan.saludar();
```

El constructor es un tipo especial de método que se llama al crear un objeto. En TypeScript, el método constructor siempre se define con el nombre "constructor".

```typescript
class Employee {

  empCode: number;
  empName: string;

  constructor(empcode: number, name: string ) {
      this.empCode = empcode;
      this.name = name;
  }
}
```

En el ejemplo anterior, la clase Employee incluye un constructor con los parámetros empcode y name. En el constructor, se puede acceder a los miembros de la clase mediante la palabra clave this, por ejemplo this.empCode, o this.name.

No es necesario que una clase tenga un constructor.

```typescript
class Employee {

  empCode: number;
  empName: string;

  constructor(empcode: number, name: string ) {
        this.empCode = empcode;
        this.empName = name;
  }
}

let emp = new Employee(100,"Steve");
```

En el ejemplo anterior, pasamos valores al objeto para inicializar las variables miembro. Cuando creamos una instancia de un nuevo objeto, se llama al constructor de la clase con los valores pasados y las variables miembro empCodey empName se inicializa con estos valores.

```typescript
class Persona {

  private _edad: number;

  constructor(edad: number) {
    this._edad = edad;
  }

  get edad() {
    return this._edad;
  }

  set edad(valor: number) {
    if(valor < 0) {
      throw new Error("La edad no puede ser negativa");
    }
    this._edad = valor;
  }

}

const persona = new Persona(25);

// Getter
console.log(persona.edad); // 25

// Setter
persona.edad = 30;
console.log(persona.edad); // 30
```

En este ejemplo:

La propiedad edad es privada () para encapsularla
El getter edad() devuelve el valor de _edad
El setter edad(valor) permite asignar nuevo valor con validación
Usamos el getter y setter para acceder y modificar el valor de forma controlada
Así se utilizan los getters y setters en TypeScript para aplicar encapsulamiento y control de acceso a propiedades de clase.

Encapsulamiento - Ocultar detalles y exponer una interfaz simple. Por ejemplo, ocultar la lógica interna de métodos.

```ts
class Persona {
  nombre: string;

  constructor(nombre: string) {
    this.nombre = nombre;
  }

  caminar() {
    console.log(`${this.nombre} está caminando.`);
  }

}

class Estudiante extends Persona {

  asignaturas: string[];

  constructor(nombre: string, asignaturas: string[]) {
    super(nombre);
    this.asignaturas = asignaturas;
  }

  estudiar() {
    console.log(`${this.nombre} está estudiando ${this.asignaturas}.`)
  }

}
const estudiante = new Estudiante("Juan", ["Matemáticas", "Historia"]);
estudiante.caminar(); // Juan está caminando.
estudiante.estudiar(); // Juan está estudiando Matemáticas,Historia.
```

La clase Estudiante extiende Persona con la palabra extends
Estudiante hereda el constructor y método caminar() de Persona
Además tiene su propiedad asignaturas y método estudiar()
Se utiliza super() para llamar al constructor de la clase padre

Ejemplo interfaces - implementación

```ts
interface Machine {
  type: string;
}

interface Vehicle {
  wheels: number;
}

class Car implements Machine, Vehicle {
  private _type: string;
  private _wheels: number;

  constructor(type: string, wheels: number) {
    this._type = type;
    this._wheels = wheels;
  }

  get type() {
    return this._type;
  }

  set type(value) {
    this._type = value;
  }

  get wheels() {
    return this._wheels;
  }

  set wheels(value) {
    this._wheels = value;
  }
}

const car = new Car("sedan", 4);
car.type = "hatchback";
car.wheels = 2;
console.log(car);
```

```typescript
interface IVehicle {
  wheels: number;
  drive(): void;
}

class Vehicle {
   wheels: number;

  constructor(wheels: number) {
    this.wheels = wheels;
  }
}

class Car extends Vehicle implements IVehicle {

  constructor() {
    super(4);
  }

  drive() {
    console.log(`El auto se mueve con ${this.wheels} ruedas`);
  }
}

const car = new Car();
car.drive();
```

Ejemplo interface con herencia de clases

Ejercicio para realizar en TypeScript.

1. Crear una clase Persona con propiedades como nombre, edad, documento identidad y métodos como caminar(), hablar(), comer(). Crear algunos objetos Persona e interactuar con sus propiedades y métodos.
2. Crear una clase CuentaBancaria con propiedades como número de cuenta, titular y saldo. Agregar métodos para depositar, retirar y consultar saldo. Crear varias cuentas y probar los métodos.
3. Crear una clase Vehículo con subclases Coche, Barco y Avión. Cada subclase implementa su propio método desplazarse (). Crear objetos de las subclases y probar sus métodos.
4. Crear una clase FiguraGeometrica y subclases Triángulo, Círculo y Cuadrado. Implementar el método area() en cada subclase. Crear objetos y calcular sus áreas.
5. Crear una clase Electrodoméstico con subclases Televisor, Nevera y Lavadora. Cada electrodoméstico tiene propiedades como precio y color. Crear algunos objetos y probar.
6. Crear una clase Hotel con propiedades como nombre y ubicación. Crear clase Habitación con número de habitación, precio y estado. Agregar métodos para reservar y liberar habitación. Probar con algunos hoteles e interacciones.
7. Crear una clase Película con propiedades como título, duración y director. Crear clase CatalogoPeliculas para almacenar películas en una lista. Agregar búsqueda por título y filtrado por director. Probar con un catálogo de películas.