Interfaces Vs. Clases

1. Cread una Interfaz o Clase que se llame Producto y que contenga: Nombre, Precio y Oferta (opcional). En el programa principal, tendréis que crear los siguientes productos dentro de un array, crear una función que muestre todos los productos en oferta y otra que los muestre todos ordenenados de mayor a menor precio:
   - Camisa de 60€.
   - Pantalones de 45€, además estará en oferta.
   - Polo de 35€.
   - Slip de 15€, ademas estará de oferta.

Función 1:

```
1   interface Producto {
2       Nombre: string;
3       Precio: number;
4       Oferta?: boolean;
5   }
6
7   let productos: Producto[] = [{Nombre: 'Camisa', Precio: 60},
8                                {Nombre: 'Pantalones', Precio: 45, Oferta: true},
9                                {Nombre: 'Polo ', Precio: 35},
10                               {Nombre: 'Slip ', Precio: 15, Oferta: true}];
11
12  // función anonima para mostrar productos en oferta
13  productos.forEach(function (producto): void {
14      if(producto.Oferta){
15          console.log(`El producto es ${producto.Nombre}, y su precio ${producto.Precio}€`);
16      }
17  });
18
```

```
[LOG]: "El producto es
Pantalones, y su precio 45€"
---
[LOG]: "El producto es Slip , y
su precio 15€"
```

Función 2:

```
interface Producto {
    Nombre: string;
    Precio: number;
    Oferta?: boolean;
}

let productos: Producto[] = [{Nombre: 'Camisa', Precio: 60},
                             {Nombre: 'Pantalones', Precio: 45, Oferta: true},
                             {Nombre: 'Polo ', Precio: 35},
                             {Nombre: 'Slip ', Precio: 15, Oferta: true}];

productos.sort((b, a) => a.Precio - b.Precio);

// función anonima para mostrar los productos
productos.forEach(function (producto): void {
    console.log(`El producto es ${producto.Nombre}, y su precio ${producto.Precio}€`);
});

```

```
[LOG]: "El producto es Camisa, y su precio 60€"
[LOG]: "El producto es Pantalones, y su precio 45€"
[LOG]: "El producto es Polo , y su precio 35€"
[LOG]: "El producto es Slip , y su precio 15€"
```

2. A partir de la documentación de este enlace *"Links"*. Cread una Interfaz o Clase que se llame Persona y que contenga: Nombre, Edad, DNI y Domicilio. Tiene que tener constructor con parámetros en el que se inicialicen por defecto todos los parámetros y cada parámetro tiene que tener getter + setter. Además tiene que tener una función que se llame MostrarInfo, que devuelva toda la información de la persona en una cadena de texto. En el programa principal, tendréis que crear 2 usuarios de prueba y mostrar su información por consola.

```
class Persona {

    // Valores por defecto
    constructor(
        private Nombre = "",
        private Edad = 0,
        private DNI = "",
        private Domicilio = "") {
        this.Nombre = "Edgar";
        this.Edad = 22;
        this.DNI = "47983846G";
        this.Domicilio = "Carrer Sant Pere 54";
    }
```

```
    // Getters and setters
    get nombre() {
        return this.Nombre;
    }
    set nombre(value) {
        this.Nombre = value;
    }
    // ---------------------
    get edad() {
        return this.Edad;
    }
    set edad(value) {
        this.Edad = value;
    }
```

```
    // ---------------------
    get dni() {
        return this.DNI;
    }
    set dni(value) {
        this.DNI = value;
    }
    // ---------------------
    get domicilio() {
        return this.Domicilio;
    }
    set domicilio(value) {
        this.Domicilio = value;
    }
```

```
    // Funciones
    MostrarInfo(): string {
        return `El nombre de la persona es ${this.Nombre}, su edad es ${this.Edad}, su DNI ${this.DNI}
        y su domicilio es: ${this.Domicilio}`
    }
}

let p1 = new Persona("Ivan", 18, "457234845H", "Andorra");

console.log(p1.MostrarInfo());
```

3. Cread una Clase que se llame **Cafetera** y que contenga los : Capacidad Maxima (la cantidad máxima de café que puede contener la cafetera en ml) y Cantidad Actual (la cantidad actual de café que hay en la cafetera en ml). Tiene que tener almenos las siguientes funciones:

Constructor por defecto que establezca la capacidad máxima en 1000 y la actual en cero (cafetera vacía).

```
class Cafetera {

    // Valores por defecto
    constructor(
        private capacidadMaxima = 1000,
        private cantidadActual = 0,){
        this.capacidadMaxima;
        this.cantidadActual;
    }

```

Los getters y setters que correspondan.

```
    // Getters and setters
    get capacidadMax() {
        return this.capacidadMaxima;
    }
    set capacidadMax(value) {
        this.capacidadMaxima = value;
    }
    // ---------------------
    get cantidadAct() {
        return this.cantidadActual;
    }
    set cantidadAct(value) {
        this.cantidadActual = value;
    }
```

Función **Llenar Cafetera**: Hace que la cantidad actual sea igual a la capacidad.

```
    // Funciones
    // Llena por completo la cafetera ------------
    llenarCafetera(): void {

       this.cantidadActual = this.capacidadMaxima;
    }
```

Función **Servir Taza**: Simula la acción de servir una taza con la capacidad que se le pase por parámetro. Si la cantidad actual de café “no alcanza” para llenar la taza, se sirve lo que haya en la cafetera.

```
32      // ----------------------------
33      servirTaza(taza: number): void {
34
35          this.cantidadAct -= taza;
36
37          if (this.cantidadAct < 0) {
38              this.cantidadAct = 0;
39              console.log(`Se ha servido todo el cafe que se podia`)
40          } else {
41              console.log(`Se ha servido el cafe especificado y se ha reducido los
42                          litros que tiene la cafetera`)
43          }
44
45      }
```

```
68  let cafetera = new Cafetera();
69  let tazaDeCafe = 30;
70
71  cafetera.servirTaza(tazaDeCafe)
```

Función **Vaciar Cafetera**: Pone la cantidad de café actual a cero.

```
46      // Resetea la cafetera -------
47      vaciarCafetera(): void {
48
49          this.cantidadActual = 0;
50      }
```

Función **Agregar Café**: Añade a la cafetera la cantidad de café indicada por parámetro.

```
52      // Agregar Cafe ------------------------------
53      agregarCafe(agreg: number): void {
54
55          this.cantidadAct += agreg;
56          console.log(`Se ha añadido el cafe`)
57      }
```

```
60  let cafetera = new Cafetera();
61  let tazaDeCafe = 30;
62  let cafeAgregado = 20;
63
64  cafetera.servirTaza(cafeAgregado)
```

4. Se plantea desarrollar un programa que permita la gestión de una empresa agroalimentaria que trabaja con tres tipos de productos: Frescos, Refrigerados y Congelados. Todos los productos llevan esta información en común: Nombre del producto y Número de lote. A su vez, cada tipo de producto lleva alguna información específica:
   - Los productos frescos deben llevar la fecha de envasado y el país de origen.
   - Los productos refrigerados deben llevar el código del organismo de supervisión alimentaria.
   - Los productos congelados deben llevar la temperatura de congelación recomendada.

   Implementa las clases Producto, ProductoFresco, ProductoRefrigerado y ProductoCongelado. Tenéis que usar una relación de herencia entre las clases que pertoca. Cada clase debe disponer de constructor y permitir establecer (set) y recuperar (get) el valor de sus atributos y tener un método que permita mostrar la información del objeto.

```typescript
interface productos{
    nombreProducto: string;
    nLote: number;
}

```

```typescript
class Frescos implements productos{
    nombreProducto: string = "";
    nLote: number = 0;

    constructor(
    private fechaEnvasado: number = 1012023,
    private paisOrigen: string = "España"){
        this.fechaEnvasado;
        this.paisOrigen;
    }

    // Getters and setters
    get fechaEnvas() {
        return this.fechaEnvasado;
    }
    set fechaEnvas(value) {
        this.fechaEnvasado = value;
    }
```

```
class Congelados implements productos{
    nombreProducto: string = "";
    nLote: number = 0;

    constructor(
    private tempCongelacion: number = -18){
        this.tempCongelacion;
    }

    // Getters and setters
    get temp() {
        return this.tempCongelacion;
    }
    set temp(value) {
        this.tempCongelacion = value;
    }
}
```

```
class Refrigerados implements productos{
    nombreProducto: string = "";
    nLote: number = 0;

    constructor(
    private cosa: string = "L401319S4140"){
        this.cosa;
    }

    // Getters and setters
    get cfet() {
        return this.cosa;
    }
    set cfet(value) {
        this.cosa = value;
    }
}
```