4. Classes

Typescript

Definció d’una classe

```
class Person {
    name: string = '';
    constructor(name : string){
        this.name = name;
     }
}

const person = new Person();
person.name = "Jane";
```

Getters i Setters

En JS quan una variable és d'ús privat es posa un _ per indicar al usuari de l'objecte que són atributs interns i que no s'han de fer servir.

Podem definir el getter d'una propietat fent servir la paraula reservada `get` i el nom de la propietat com una funció que retorna un valor. Igual passa amb `set`.

```
class User {
    _name: string = ''
    _email: string = ''

    constructor (name: string, email: string){
        this._name = name
        this._email = email
    }

    get email() : string {
        return this._email
    }
    set email(email: string){
        this._email  = email
    }
}
let u = new User('pepito', 'pepito@gmail.com')
```

Exportació de classes

Fent servir l'exportació de mòduls podem també exportar la classe per fer-la servir en altres fitxers. Fent servir `export class` podem fer-la visible fora del nostre mòdul.

```
//User.ts
export default class User {
   name: string = ''
   email: string = ''
   constructor (name: string, email: string){
       this.name = name
       this.email = email
   }
}
```

```
//app.ts
import User from "./model/User";
let u = new User('pep', 'pep58@gmail.com')
```

Sobrecàrrega de mètodes

Com a les funcions podem definir les signatures els mètodes d'una classe. Limitant la manera en les que es pot cridar aquests mètodes.

```
class User {
    name: string = ''
    salutacio() : string;
    salutacio(value : HTMLElement) : string;
    salutacio(text : string | HTMLElement = 'Hello') : string{
        if(typeof text === 'string'){
            return `${text} ${this.name}`
        }else{
            return `${text.innerHTML} ${this.name}`
        }
    }
}
let u = new User()
u.salutacio()
u.salutacio("Hola")
u.salutacio(document.getElementById('salutacio'))
```

Visibilitat d’atributs

Hi ha tres modificadors de visibilitat principals a TypeScript.

- **public**: (per defecte) permet l'accés al membre de la classe des de qualsevol lloc.
- **private**: només permet l'accés al membre de la classe des de la classe
- **protected**: permet l'accés al membre de la classe des d'ell mateix i de qualsevol classe que l'hereti, que es tracta a la secció d'herència següent

Visibilitat d’atributs

```
class User {
    private name: string = ''
    private email: string = ''
    public constructor (name: string, email: string){
        this.name = name
        this.Email = email
    }
    public get Email() : string {
        return this.email
    }
    protected set Email(email: string){
        this.email  = email
    }
    public salutacio(text : string = 'Hello'){
        return `${text} ${this.name}`
    }
}
```

Exercicis

1. Crea una classe Conductor amb les propietats nom, data de naixement i un booleà que indica si pot portar moto. Crea els getters i setters per a cada propietat i si la data de naixement és inferior a 18 anys que generi un error. Fes servir la visibilitat d'atributs.
2. A la classe Conductor afegeix un mètode `print()` que retorni un text amb tota la informació del conductor. Fes que la funció `print()` pugui acceptar un paràmetre de tipus text, date o booleà i modifiqui el valor que retorna pel passat per paràmetre. Ex: 'Nom: Pepito, Edat: 32, Pot portar moto: Sí' print(false) -> 'Nom: Pepito, Edat: 32, Pot portar moto: No'

Classes genèriques

Una classe genèrica en TypeScript és una plantilla de classe que permet treballar amb diferents tipus de dades sense perdre la seguretat de tipus.

```
class Coleccio<T> {
   private elements: T[] = [];
    afegir(element: T): void {
     this.elements.push(element);
   }
    obtenir(index: number): T {
     return this.elements[index];
   }
 }

let c = new Coleccio<string>()
c.afegir('Hola')
let txt : string = c.obtenir(0)
```

Exercicis

TODO exercicis classes genèriques

Herència

Fent servir la paraula reservada `extends` es pot heretar una classe base. D'aquesta manera la classe filla adquireix les propietats i mètodes de la classe base.

```
class Fill extends Base
```

Podem fer crides al constructor, atributs i mètodes de la classe base fent servir la paraula reservada `super`.

```
class Animal {
    name: string;
    constructor(name: string) {
        this.name = name;
    }
    move(distance: number = 0) {
        console.log(`${this.name} moved ${distance}m.`);
    }
}
class Snake extends Animal {
    constructor(name: string) {
        super(name);
    }
    move() {
        super.move(5);
    }
}
```

Interfícies

Una interfície és un contracte on totes les classes que la implementen compleixen amb aquest contracte. Dins d'aquest contracte podem definir propietats i funcions que han d'implementar les classes que implementen la interfície. Per indicar que una classe implementa una interfície fem servir:

```
NomClasse implements NomInterficie
```

```
interface IVehicle {
    marca: string;
    accelerar(): void;
    frenar(): void;
}

class Cotxe implements IVehicle {
    marca: string = '';
    rodes: number = 4;
    accelerar() {
        console.log('Cotxe accelera')
    }
    frenar() {
        console.log('Cotxe accelera')
    }
}
```

Exercicis

1. Implementa la classe Rectangle on té les propietats base i alçada. Crea la classe Quadrat que hereta de Rectangle.
2. Implementa la classe `Cercle` que té la propietat radi. Crea la interfície `Perimetrable` que té la funció `perimetre()` : number que ha de retornar el perímetre de les classes que implementen. Fes que Cercle i Rectangle implementin la interfície `Perimetrable`.