Le classi in TypeScript

Docente
Mario Perna
prof.perna.mario@darzo.net

A.S.
2025/2026
Materia
TEPSIT (Laboratorio)

Introduzione

In TypeScript, come in molti altri linguaggi di programmazione orientati agli oggetti, le classi e gli oggetti sono fondamentali per organizzare e strutturare il codice in modo più efficace ed efficiente.

Le classi rappresentano le strutture logiche (attributi e metodi) degli oggetti, i quali rappresentano istanze specifiche di tali classi.

La programmazione ad oggetti

Nella programmazione orientata agli oggetti, è possibile utilizzare l'ereditarietà per creare nuove classi basate su classi esistenti, ereditando proprietà e metodi.

TypeScript fornisce anche livelli di accesso, come **public**, **private** e **protected**, per controllare l'accesso ai membri delle classi e per garantire l'incapsulamento dei dati.

Precisazioni sulle classi in TS

Da JavaScript abbiamo compreso che gli oggetti sono delle entità autonome e non derivano da classi specifiche.

TypeScript prevede il supporto delle classi rendendolo più vicino alla programmazione ad oggetti classica (es. Java).

Dichiarazione di una classe

Le parole chiave per la definizione di una classe è **class** seguita dal nome della classe (per convenzione con la prima lettera maiuscola).

Con la parola chiave **constructor** viene definito il metodo costruttore (default o parametrizzato)

Dichiarazione di una classe

```
1  //Definzione della classe persona
2  class Persona {
3      //Attributi della classe
4      private nome: string;
5      private età: number;
6      //Metodo costruttore 'parametrizzato'
7      constructor(nome: string, età: number) {
8          this.nome = nome;
9          this.età = età;
10     }
11     //Metodo della classe
12     saluta(): void {
13         document.write(`Ciao, sono ${this.nome} e ho ${this.età} anni.`);
14     }
15 }
```

Dichiarazione e istanza di una oggetto

Con la parola chiave **new** avviene la fase di istanza dell'oggetto, a seguire il metodo costruttore. Con la notazione "**.**" vengono invocati i metodi della classe.

```
17  const persona1: Persona = new Persona("Harry", 25);
18  const persona2: Persona = new Persona("Hermione", 24);
19
20  persona1.saluta(); // Output: Ciao, sono Harry e ho 25 anni.
21  persona2.saluta(); // Output: Ciao, sono Hermione e ho 24 anni.
```

Ereditarietà

Con **extends** viene ereditata la struttura logica della classe madre.

Con **this** viene invocato l'attributo della classe corrente.

Con **super** viene gestita la relazione tra la classe madre e classi figlie. È utile per chiamare il costruttore della classe madre.

Ereditarietà

```
23  class Studente extends Persona {
24      private matricola: number;
25
26      constructor(nome: string, età: number, matricola: number) {
27          super(nome, età);
28          this.matricola = matricola;
29      }
30
31      studia() {
32          document.write(`${this.nome} sta studiando.`);
33      }
34  }
```

Ereditarietà

```
37  const studente1: Studente = new Studente("Harry Potter", 22, 12345);
38  studente1.saluta(); // Output: Ciao, sono Harry Potter e ho 22 anni.
39  studente1.studia(); // Output: Harry Potter sta studiando.
```

Nota: nell'esempio, ho modificato le righe **4 e 5** della slide nr. **6** cambiando il livello di visibilità da **private** a **protected** in modo da poter accedere alle proprietà della classe **Persona** dalla sottoclasse **Studente**.

Livelli di visibilità

TypeScript supporta i livelli di visibilità: public, private, protected.

- **public**: accessibile a tutte le classi.
- **private**: accessibile solo all'interno della classe.
- **protected**: accessibile all'interno della classe e delle sottoclassi.

NOTA: **il livello di visibilità di implicito (o di default) è public.**

Metodi setter e getter

Con **set** è possibile definire metodi per assegnare valori ai singoli attributi, e non solo…

Con **get** è possibile ottenere i dati contenuti negli attributi dei singoli oggetti.

```
class Studente{
    private nome: string;
    private cognome: string;
    private matricola: string;

    constructor(){
        this.nome = "";
        this.cognome = "";
        this.matricola = "";
    }
    //Metodo setter
    set setNome(n: string) {
        this.nome = n;
    }
    //Metodo getter
    get getNome(): string{
        return this.nome;
    }
}
```

I membri statici

Un membro statico è una proprietà o un metodo che possono essere utilizzati senza necessità di creare un'istanza della classe.

```
1  class Persona {
2      nome: string;
3      cognome: string;
4      constructor(nome, cognome) {
5          this.nome = nome;
6          this.cognome = cognome;
7      }
8      static concatena(a:string, b:string) {
9          return a + " " + b;
10     }
11 }
12
13 document.write(Persona.concatena("Harry", "Potter"));  //Harry Potter
```

Sitografia

https://profsiciliano.altervista.org
https://www.typescriptlang.org