

Ereditarietà


Ereditarietà di prototipi

Quando si accede in lettura a una proprietà, se la chiave non è definita nell'oggetto, si va a cercare nel prototipo (e via così, ricorsivamente).

È anche possibile **impostare** il prototipo di una nuova classe a un'altra classe.

Grazie al meccanismo dei prototipi, gli oggetti della nuova classe avranno anche tutti i metodi definiti dalla classe che viene **estesa**. In più, la sottoclasse può aggiungere ulteriori metodi.

Solo le **istanze** della **sottoclasse** avranno i metodi aggiunti.



```
class Studente extends Persona {  
  laurea() {  
    return "Evviva!"  
  }  
}  
  
var ugo = new Studente("Ugo",19)  
  
ugo → Studente { nome: 'Ugo', 'età': 19 }  
ugo.compleanno() →  
ugo → Studente { nome: 'Ugo', 'età': 20 }  
ugo.laurea() → Evviva!  
  
pippo.laurea() → TypeError: pippo.laurea is  
not a function
```

Applicando il meccanismo di ricerca nella catena dei prototipi, osserviamo che:

- Una sottoclasse può **ridefinire** un metodo già definito in una sua superclasse (diretta o indiretta)
- Una sottoclasse eredita tutti i metodi della superclasse che non ridefinisce
- Usando l'operatore **delete** (sul prototipo) è anche possibile cancellare dalla sottoclasse un metodo definito dalla superclasse senza sovrascriverlo
- Un'istanza può cancellare o sovrascrivere un metodo della propria classe

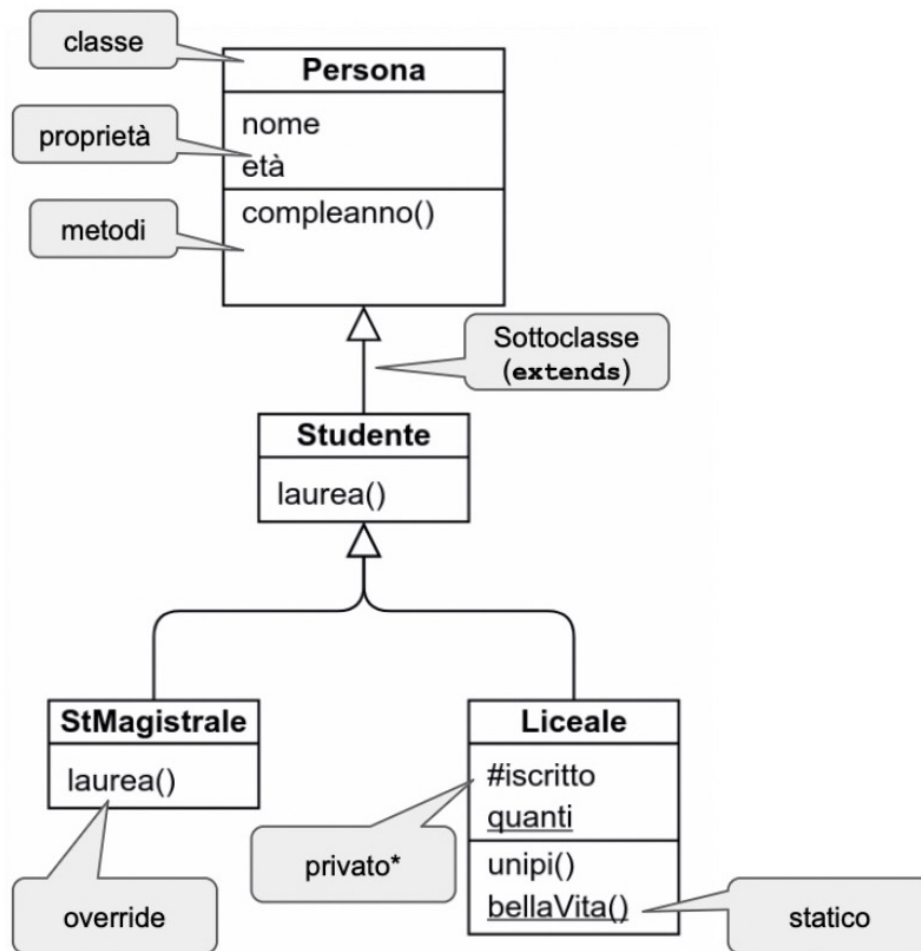
Capita spesso che il codice in una sottoclasse debba fare riferimento ai metodi della sua superclasse (per esempio, per completarli o aggiungere caratteristiche).

La parola chiave `super` è un riferimento alla superclasse di un oggetto, come `this` è un riferimento all'oggetto su cui il metodo è invocato.



Una classe è in realtà una funzione, quindi `super` è una funzione

```
class StMagistrale extends Studente {
  constructor(nome, età, triennale) {
    super(nome,età)
    this.triennale=triennale
  }
  laurea() {
    return super.laurea()+" Evviva galattico!"
  }
}
```



Getter

È un metodo che viene chiamato quando si tenta di accedere al valore di una proprietà di un oggetto. In altre parole, quando si accede alla proprietà tramite la sintassi "nomeOggetto.nomeProprietà", il getter viene chiamato per recuperare il valore della proprietà.

Ad esempio:

```
var persona = {  
  nome: "Mario",  
  cognome: "Rossi",  
}
```

```
get nomeCompleto() {  
    return this.nome + " " + this.cognome;  
}  
};  
  
console.log(persona.nomeCompleto); // output: "Mario Rossi"
```

Quando si vuole leggere il valore di una proprietà di un oggetto, si guarda se l'oggetto ha la chiave cercata.

1. Se la chiave è presente, si controlla se è un getter o una proprietà base
 - a. Se è un getter, si invoca la funzione corrispondente, e il valore è quello restituito dalla funzione
 - b. Se è una proprietà base, il valore è quello della chiave nell'oggetto
2. Se la chiave non è presente, e l'oggetto ha un prototipo, si cerca la proprietà nel prototipo, ricorsivamente
3. Se la chiave non è presente, e l'oggetto non ha un prototipo, il risultato è **undefined**

Setter

È un metodo che viene chiamato quando si tenta di assegnare un valore a una proprietà di un oggetto. In altre parole, quando si assegna un valore alla proprietà tramite la sintassi "nomeOggetto.nomeProprietà = valore", il setter viene chiamato per impostare il nuovo valore della proprietà.

Ad esempio:

```
var persona = {  
    nome: "Mario",  
    cognome: "Rossi",  
    set nomeCompleto(value) {  
        var parti = value.split(" ");  
        this.nome = parti[0];  
        this.cognome = parti[1];  
    }  
}
```

```
};  
  
persona.nomeCompleto = "Luigi Verdi";  
console.log(persona.nome); // output: "Luigi"  
console.log(persona.cognome); // output: "Verdi"
```

Quando si vuole scrivere il valore di una proprietà di un oggetto, si guarda se l'oggetto ha la chiave cercata.

1. Se la chiave è presente, si controlla se è un setter o una proprietà base
 - a. Se è un setter, si invoca la funzione corrispondente, passando come unico argomento il valore che si vuole assegnare
 - b. Se è una proprietà base, si assegna il valore alla proprietà (eventualmente sovrascrivendo il valore precedente)
2. Se la chiave non è presente, si cerca ricorsivamente un setter nel prototipo, ricorsivamente
 - a. Se si trova un setter lungo la catena, si invoca la funzione corrispondente, passando come unico argomento il valore che si vuole assegnare
 - b. Se non si trova un setter lungo la catena, la proprietà viene aggiunta all'oggetto, con il valore che si vuole assegnare.

Le parole chiave `get` e `set`, davanti al nome della proprietà, creano i metodi di accesso. Dall'esterno, la proprietà appare come una normale chiave con valore.

Ogni lettura di proprietà causa l'invocazione del getter. Ogni scrittura di proprietà causa l'invocazione del setter (passando come argomento il valore assegnato).



I metodi di accesso possono essere usati, per esempio, per consentire l'accesso allo stesso dato in modi diversi.

Generatori

Sono un particolare tipo di funzione (e dunque di metodo)

I generatori si dichiarano con `function* f() {}` anziché `function f() {}`, oppure `*metodo() {}` anziché `metodo() {}` (dentro le classi).

Nel corpo di un generatore (e solo lì) si può usare il comando **yield**, che restituisce al chiamante il valore di espressione, ma riprende l'esecuzione dal comando successivo (e non dall'inizio del corpo) in caso di "rientro"

Il generatore è un oggetto con due proprietà: **value** è il valore restituito, e **done** vale true se la generazione è completa.

Nella pratica, spesso lo scorrimento è fatto con un `for (... of ...)`, e non si invoca esplicitamente `next()`.