2022_03_31

April 1, 2022

1 2022-03-31

1.0.1 Corso ITS

1.1 Magento & e-commerce software

1.2 ### JavaScript (Andrea Ribuoli)

1.3 # iiiii

1.4 In JavaScript le classi usano un'ereditarietà basata su prototipi

1.4.1 cioè:

1.5 se due oggetti ereditano proprietà dallo stesso prototipo

2 allora

2.1 quegli oggetti sono istanze della stessa classe

2.1.1 Quando creo un oggetto creo una istanza, non una vera e propria classe (tipo)

2.1.2 Con ECMAScript 2015 (ES6) è stata introdotta la keyword class

2.2 ma non è cambiata la sostanza!

2.2.1 Tale nuova keyword (class) favorisce l'ereditarietà tra... oggetti

2.3 ## non classi (!) che, a rigore, non esistono in JavaScript

3 Dobbiamo fare un sacco di distinguo...

4 In genere. ma non necessariamente, se due oggetti ereditano dallo stesso prototipo

5 significa che sono stati creati e inizializzati dalla stessa funzione costruttore

[34]:
```
!echo 'console.log("Ciao a tutti!")' | node -
```

```
Ciao a tutti!
```

[35]:
```
!echo 'console.log( \
"Ciao!")' | node -
```

```
Ciao!
```

[36]:
```
!echo 'console.log( \
"Ciao!")'
```

```
console.log(  "Ciao!")
```

[38]:
```
!echo 'console.log("Ciao!"); \
console.log("Ciao!")'
```

```
console.log("Ciao!");  console.log("Ciao!")
```

[37]:
```
!echo 'console.log("Ciao!"); \
console.log("Ciao!")' | node -
```

```
Ciao!
Ciao!
```

[42]:
```
!echo 'function utente(nome) { \
         let u = Object.create(null); \
         u.nome = nome; \
         return u; }; \
       let a = utente("Andrea"); \
       console.log(a);' | node -
```

```
[Object: null prototype] { nome: 'Andrea' }
```

[39]:
```
!echo 'function utente(nome) { \
         let u = Object.create(utente.methods); \
         u.nome = nome; \
         return u; }; \
       utente.methods = { \
         toString() { return "Utente: " + this.nome; } \
       }; \
       let a = utente("Andrea"); \
       console.log(a.toString()); \
       console.log(a);' | node -
```

```
Utente: Andrea
{ nome: 'Andrea' }
```

[43]:
```
!echo 'function utente(nome, cognome) { \
         let u = Object.create(utente.methods); \
         u.nome = nome; \
         u.cognome = cognome; \
         return u; }; \
       utente.methods = { \
```

```
        toString() { return "Utente: " + this.nome \
                                + " " + this.cognome; } \
      }; \
      let a = utente("Andrea", "Ribuoli"); \
      console.log(a.toString()); \
      console.log(a);' | node -
```

```
Utente: Andrea Ribuoli
{ nome: 'Andrea', cognome: 'Ribuoli' }
```

```
function utente(nome) {
  let u = Object.create(utente.methods)
  u.nome = nome
  return u }
utente.methods = {
  toString() { return "Utente: " + this.nome }
};
let a = utente("Andrea")
console.log(a.toString())
console.log(a)
```

6 parliamo di funzione factory

6.1 utilizziamo la proprietà *methods* della funzione stessa

6.2 e questa proprietà memorizza l'oggetto prototipo utilizzato nella Object.create

6.3 La proprietà nome sarà specifica di ogni singolo oggetto *utente*

7 non è vincolante il nome methods ma è una buona convenzione

[44]:
```
!echo 'function utente(nome) { \
        let u = Object.create(utente.metodi); \
        u.nome = nome; \
        return u; }; \
      utente.metodi = { \
        toString() { return "Utente: " + this.nome; } \
      }; \
      let a = utente("Andrea"); \
      console.log(a.toString()); \
      console.log(a);' | node -
```

```
Utente: Andrea
{ nome: 'Andrea' }
```

7.1 La notazione per dichiarare le funzioni è quella compatta (ES6)

8 Cosa significa this?

8.1 fa riferimento all'oggetto attraverso il quale la funzione stessa è stata invocata

8.2 quindi *this.nome* sarà la proprietà *nome* dell'istanza (oggetto) *a* dell'esempio

9 Abbiamo visto un metodo semplice ma non idiomatico del mondo object-oriented

[31]:
```
!echo 'function Utente(nome) { \
        this.nome = nome; \
      }; \
      Utente.prototype = { \
        toString: function() { return "Utente: " + this.nome; } \
      }; \
      let a = new Utente("Andrea"); \
      console.log(a.toString()); \
      console.log(a);' | node -
```

```
Utente: Andrea
{ nome: 'Andrea' }
```

10 L'uso del maiuscolo (*Utente* anzichè *utente*) è una convenzione, ma importante

11 Il costruttore è invocato con la *parola riservata new*

```
function Utente(nome) {
    this.nome = nome
}
Utente.prototype = {
    toString: function() { return "Utente: " + this.nome }
}
let a = new Utente("Andrea")
console.log(a.toString())
console.log(a)
```

[2]:
```
!echo 'function Utente(nome) { \
        this.nome = nome; \
      }; \
      Utente.prototype = { \
        toString: function() { return "Utente: " + this.nome; } \
      }; \
      let a = new Utente("Andrea"); \
      console.log(a.toString()); \
      console.log(a)' | node -
```

```
Utente: Andrea
{ nome: 'Andrea' }
```

12 la parola riservata class

```
class Utente {
  constructor(nome) {
    this.nome = nome
  }
  toString() { return "Utente: " + this.nome }
}
let a = new Utente("Andrea")
console.log(a.toString())
console.log(a)
```

[3]:
```
!echo 'class Utente { \
         constructor(nome) { \
           this.nome = nome; \
         }; \
         toString() { return "Utente: " + this.nome; } \
       }; \
       let a = new Utente("Andrea"); \
       console.log(a.toString()); \
       console.log(a)' | node -
```

```
Utente: Andrea
Utente { nome: 'Andrea' }
```