

Corso Enaip
Rimini 06/03/2023

Agenda

- Ereditarietà
- Funzioni avanzate e le clousure
- Le arrows function
- I metodi delle classi Array, String
- Il tipo Date

Ereditarietà

Ereditarietà

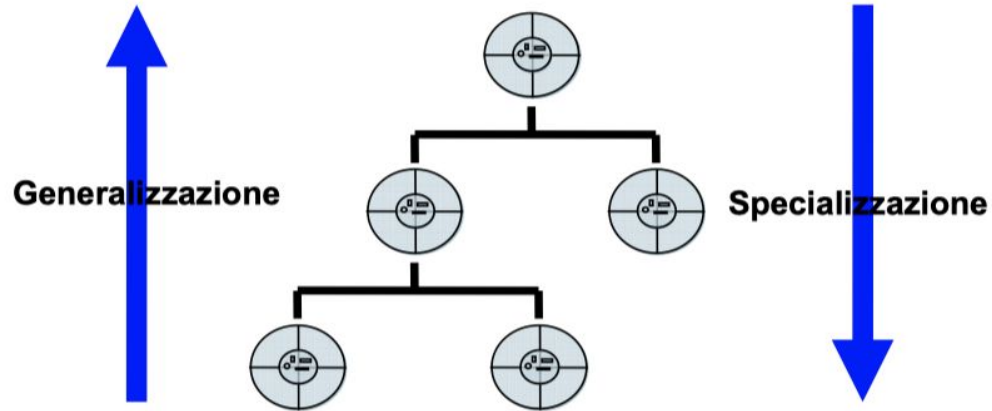
- L'ereditarietà consente di **definire nuove classi per specializzazione o estensione di classi preesistenti, in modo incrementale**
- L'ereditarietà è una caratteristica che permette ad **una classe di estendere le proprietà di altre classi.**

Ereditarietà

- Il meccanismo dell'ereditarietà è di **fondamentale importanza** nella programmazione ad oggetti, in quanto **induce** una **strutturazione gerarchica** nel sistema software da costruire.
- L'ereditarietà consente infatti di realizzare **relazioni** tra classi di tipo **generalizzazione-specializzazione**, in cui una classe, detta base, realizza un **comportamento generale comune** ad un insieme di entità, mentre le classi derivate (sottoclassi) **realizzano comportamenti specializzati** rispetto a quelli della classe base.
 - Esempio
 - Tipo o classe base: Animale
 - Tipi derivati (sottoclassi): Cane, Gatto e Etc..
- In una gerarchia gen-spec, le classi derivate sono specializzazioni (cioè casi particolari) della classe base.

Ereditarietà

- **Generalizzazione:** dal particolare al generale
- **Specializzazione:** dal generale al particolare




Ereditarietà

```
class Mammifero {
  presentati () {
    console.log("Ciao io sono un " + this.tipo );
  }
}

class Gatto extends Mammifero {
  constructor () {
    this.verso = "miagolo";
    this.tipo = "gatto";
  }
  miagola () {
    console.log("Io " + this.verso + ": MEEEE0000W");
  }
}

class Cane extends Mammifero {
  constructor () {
    this.verso = "abbaio";
    this.tipo = "cane";
  }
  abbaia () {
    console.log("Io " + this.verso + ": WOOF WOOF");
  }
}
```

Sovrascrivere un metodo

```
class Gatto extends Mammifero {  
    constructor () {  
        this.verso = "miagolo";  
        this.tipo = "gatto";  
    }  
    miagola () {  
        console.log("Io " +this.verso + ": MEEEE0000W");  
    }  
  
    presentati () {  
        // ...questo verrà utilizzato per gatto.presentati()  
        // piuttosto di presentati() dal padre, class Mammifero  
          
    }  
}
```


super

Generalmente non vogliamo “rimpiazzare” completamente il metodo ereditato, ma piuttosto modificarlo leggermente o estendere le sue funzionalità.

Le classi forniscono la parola chiave "**super**" per questo scopo.

- **super.method(...)** per richiamare un metodo dal padre;
- **super(...)** per richiamare il costruttore del padre (valido solo all'interno del nostro costruttore).

Sovrascrivere un metodo

```
class Gatto extends Mammifero {  
  constructor () {  
    this.verso = "miagolo";  
    this.tipo = "gatto";  
  }  
  miagola () {  
    console.log("Io " +this.verso + ": MEEEE0000W");  
  }  
  
  presentati () {  
    super.presentati();  
    console.log("Mi piacciono molto i topi.");  
  }  
}
```

Sovrascrivere il costruttore

```
class Mammifero {
  constructor(nome) {
    this.nome = nome;
  }
  presentati() {
    console.log("Ciao io sono un " + this.tipo + " e mi chiamo " + this.nome);
  }
}

class Gatto extends Mammifero {
  constructor(name) {
    super(name);
    this.verso = "miagolo";
    this.tipo = "gatto";
  }
  miagola() {
    console.log("Io " + this.verso + ": MEEEEEOOOOW");
  }
}
```

Domande?



Riepilogo

- Per estendere una classe: **class** Figlio **extends** Padre.
- Quando sovrascriviamo un costruttore:
 - Dobbiamo richiamare il costruttore del padre attraverso **super()** nel costruttore di Figlio prima di utilizzare **this**.
- Quando sovrascriviamo un metodo:
 - Possiamo usare **super.method()** in un metodo di Figlio per richiamare il metodo da Padre.

Esercitazione 1

Si chiede di definire una serie di classi indipendenti, che rappresentano delle forme geometriche piane: Triangolo, Quadrato, Rettangolo e Cerchio.

Per ciascuna classe:

Definire il costruttore con i parametri necessari a descrivere la sua forma geometrica.

Implementare la classe padre FiguraGeometrica che contiene i seguenti metodi:

- calcolaPerimetro
- calcolaArea



Funzioni avanzate e le clousure

Linguaggio Funzionale

JavaScript è un linguaggio **fortemente orientato alle funzioni**. Una **funzione** può essere creata in qualsiasi momento, copiata su una variabile o passata come argomento ad un'altra funzione ed è possibile richiamarla da qualsiasi punto del codice.

Sappiamo che una funzione può accedere alle variabili esterne, questa caratteristica viene spesso utilizzata.

Funzioni annidate

Una funzione si definisce “annidata” quando viene creata all’interno di un’altra funzione.

```
function conta() {  
    var count = 0;  
  
    return function () {  
        return count++;  
    };  
}  
  
var counter = conta();  
  
console.log(counter()); // 0  
console.log(counter()); // 1  
console.log(counter()); // 2
```

Closure

Una closure è una funzione che ricorda le sue variabili esterne ed è in grado di accedervi. In alcuni linguaggi questo non è possibile, oppure è richiesto che la funzione venga scritta in un determinato modo (ad esempio nelle prime versioni di Java). In JavaScript tutte le funzioni sono closure di natura.

Domande?



Le Arrows Function

Le Arrow functions

Un nuovo tipo di funzione introdotta dalle specifiche di ECMAScript 2015 è rappresentato dalle **arrow function**. Si tratta di funzioni anonime con una sintassi molto concisa ed alcune specifiche caratteristiche.

Le arrow functions non sono semplicemente una “*scorciatoia*” per scrivere codice più breve.

Le Arrow functions

```
//metodo tradizionale  
var somma = function(x, y) {  
    return x + y;  
};  
  
//arrow function  
var somma = (x, y) => x + y;  
  
var totale = somma(3, 2);
```

Le Arrow functions

La sintassi generale di una arrow function prevede **le parentesi tonde intorno alla lista dei parametri** e **le parentesi graffe per delimitare il corpo della funzione**.

```
//arrow function  
var somma = (x, y) => {return x + y; }  
  
var somma = (x, y) => x + y
```

Ci sono delle eccezioni

```
//arrow function  
var somma = (x, y) => {return x + y; }  
  
var somma = (x, y) => x + y;  
  
var moltiplicaPerDue = x => x * 2;  
  
var saluta = () => "Ciao";
```


Quando usare le arrow functions

Data la **sintassi compatta** delle arrow function, esse si prestano molto bene ad essere utilizzate come **callback**.

```
var numeri = [1, 2, 3];  
numeri.forEach(valore => console.log(valore));
```

Importante

L'aspetto più importante da tener presente è che la parola chiave **this** all'interno di una arrow function **non rappresenta il contesto di esecuzione a runtime** come avviene per le funzioni standard. Per questo motivo non possono essere utilizzate come Costruttori e non possono essere utilizzate per la creazione di metodi negli oggetti e nelle classi. Un'altra limitazione è che le arrow function non accedono alla variabile **arguments**.

Riepilogo

Arrow functions:

- Non possiedono **this**
- Non possiedono **arguments**
- Non possono essere invocate con **new** (Classi e Costruttore)
- Non possiedono **super**.

Domande?



Array: metodi e proprietà

array.length: restituisce il numero di elementi dell'array.

array.join(sep): restituisce una stringa ottenuta dalla concatenazione di tutti gli elementi.

array.reverse(): inverte l'ordine degli elementi usando l'indice.

array.sort(): ordina gli elementi utilizzando l'ordine lessicografico.

array.concat(): restituisce un array con gli elementi dell'array sorgente più i valori passati come parametri.

array.slice(): restituisce un array con un sottoinsieme degli elementi, dall'indice del primo argomento all'indice del secondo argomento escluso.

array.push() e **array.pop()**: consentono di trattare l'array come uno stack(pila).

array.push(): inserisce alla fine dell'array gli elementi passati come parametri e restituisce la nuova lunghezza dell'array.

array.pop(): elimina e restituisce l'ultimo elemento.

array.forEach(): itera l'array ed esegue la funzione in input..

array.unshift() e **array.shift()**: come push() e pop(), ma inseriscono/ rimuovono elementi dall'inizio dell'array.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array?retiredLocale=it

String: metodi e proprietà

“string”.length: restituisce il numero di caratteri della stringa.

“string”.charAt(): estrae il carattere alla posizione indicata

“string”.indexOf()/lastIndexOf(): restituisce la posizione della sottostringa specificata

“string”.substr(): estrae una sottostringa

“string”.split(): restituisce un array spezzando la stringa in corrispondenza di un delimitatore o regex

“string”.toUpperCase();

“string”.toLowerCase();

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String?retiredLocale=it

Date: metodi e proprietà

new Date().getFullYear(): restituisce il valore di “year” (anno).

new Date().getMonth(): restituisce il valore di “month” (mese).

new Date().getDate(): restituisce il valore di “day” (giorno).

getHours(), getMinutes(), getSeconds(), getMilliseconds(): Fornisce il valore del corrispettivo parametro.

new Date().getDay(): restituisce il giorno della settimana, da 0 (Domenica) a 6 (Sabato). Il primo giorno è sempre Domenica;

new Date().getTime(): Ritorna il timestamp della data – il numero di millisecondi trascorsi dal 1 Gennaio 1970 in UTC+0.

Date.now(): Ritorna il timestamp della data – il numero di millisecondi trascorsi dal 1 Gennaio 1970 in UTC+0.

Date.parse(str): Esegue il parse di una Stringa e la converte in un oggetto Date.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date?retiredLocale=it

Domande?



Esercitazione 2

Scrivi un programma per la gestione di una rubrica telefonica. Definisci una classe che rappresenti un contatto. L'unico vincolo che hai è di inserire almeno il nome, cognome e il numero di telefono come stringhe.

Definisci un'altra classe che rappresenti la Rubrica che implementa i metodi per le seguenti operazioni:

- Visualizzazione dell'intera lista contatti con la possibilità di scegliere il tipo di ordinamento (A->Z, Z->A)
- Inserimento di un nuovo Contatto che non sia già presente in Rubrica.
- Modifica di uno contatto passando in input l'indice dell'array
- Cancellazione di un contatto passando in input l'indice dell'array
- Ricerca passando il nome, o parte del nome, e restituendo il singolo contatto.



Prossima lezione

- Gestione degli errori
- Promises, async/await