JavaScript OOP

Per "programmazione orientata agli oggetti" (object-oriented programming) si intende un pattern di rogrammazione che si sviluppa attorno alla creazione di oggetti software, composti di determinate proprietà e abilità, e alla loro interazione reciproca. Oggi risulta essere una delle tecniche di quando è necessario dividere agevolmente il lavoro tra diversi sviluppatori.

JavaScript tuttavia può venire utilizzato per la scrittura di codice orientato agli oggetti, in quanto è possibile definire utilizzando la sua sintassi nativa le tre logiche principali dei linguaggi OOP:

-Incapsulamento

-Ereditarietà

-polimorfismo

E' possibile quindi creare un oggetto direttamente, senza aver creato precedentemente una classe, utilizzando la cosiddetta "notazione letterale". Un oggetto può contenere degli attributi, ovvero delle variabili/costanti che ne specificano le caratteristiche, e dei metodi, che ne specificano le abilità e le operazioni che può compiere. Per accedere ai valori degli attributi o per invocare i metodi, è necessario esplorare l'oggetto tramite l'utilizzo dell'operatore "." o specificare il nome della proprietà tramite parentesi quadre.

```
// struttura prototipale di JS

let person = {
    firstName: "John",
    lastName: "Doe"
}

console.log(person.firstName)
// "John"

console.log(person.hasOwnProperty("lastName"))
// true

// l'oggetto person non possiede il metodo
// "hasOwnProperty", ma lo può invocare in quanto
// gli viene "trasmesso" dal suo prototipo, Object
```

Quando tuttavia è necessario creare molteplici oggetti con la medesima struttura, la notazione letterale risulta un impedimento, in quanto costringe a copiare/incollare l'elenco delle proprietà a mano ogni volta. **Per questo esiste il Costruttore.**

```
// definiamo un costruttore che accetti due parametri
// _firstName e _lastName

const NamedPerson = function(_firstName, _lastName) {
    this.firstName = _firstName
    this.lastName = _lastName
    this.greet = function() {
        return "Hello, my name is " + this.firstName
    }
}

// creiamo l'oggetto p1 a partire dal costruttore
const p1 = new NamedPerson("Mario", "Bianchi")

console.log(p1.greet())
// "Hello, my name is Mario"

// la proprietà firstName di p1 è stata automaticamente
```

Un costruttore è una speciale funzione (per convenzione la prima lettera va specificata maiuscola) che permette di creare, tramite la sua invocazione con la keyword "new", un oggetto "precostituito" di determinati attributi/metodi. In un colpo solo vengono assegnate molteplici coppie chiave/valore, che possono essere lette o sovrascritte. I metodi descritti nel costruttore possono contenere la keyword **"this"** al fine di accedere alle proprietà assegnate all'oggetto creato

Ma un costruttore può anche spingersi oltre: durante la sua invocazione tramite "new" è possibile passare alla funzione anche dei parametri, che possono essere utilizzati dal costruttore stesso in modo da inizializzare determinate proprietà dell'oggetto appena creato. Questo permette di ottenere un oggetto non più totalmente vuoto, ma con già dei valori assegnati agli attributi/metodi di nostra scelta. L'implementazione e l'assegnazione delle proprietà vengono descritte una volta sola all'interno di NamedPerson, la funzione costruttore.

THIS

In JavaScript, la parola chiave "this" ha un comportamento diverso da quello di molti altri linguaggi di programmazione. Essa può essere usata in qualsiasi funzione, anche se non si tratta del metodo di un oggetto. Il valore di "this" viene valutato al momento dell'esecuzione della funzione. Per comprendere il valore di "this", la regola è semplice: per invocare un metodo "func" di un oggetto "obj", si utilizzerà obj.func(); durante la chiamata di func(), "this" si riferisce a obj, ovvero all'oggetto posto prima della notazione di accesso (in questo caso, il punto).

```
// definiamo un costruttore che accetti due parametri
// _firstName e _lastName

const NamedPerson = function(_firstName, _lastName) {
    this.firstName = _firstName
    this.lastName = _lastName
    this.greet = function() {
        return "Hello, my name is " + this.firstName
    }
}

// creiamo l'oggetto p1 a partire dal costruttore
const p1 = new NamedPerson("Mario", "Bianchi")

console.log(p1.greet())
// "Hello, my name is Mario"

// la proprietà firstName di p1 è stata automaticamente
```

CLASS

JavaScript , al fine di uniformare la sintassi ha introdotto la keyword “class”. La keyword “class” non è altro che un metodo alternativo per la creazione di un costruttore. Gli oggetti creati a partire da una classe vengono definiti “**istanze**” della classe. È importante sottolineare che la keyword “class” non aggiunge nulla alle funzionalità di JavaScript e non ne altera assolutamente il modello di ereditarietà; Di fatto il costrutto “class” svolge la stessa funzione di un costruttore CON SINTASSI DIVERSA.

```
// definiamo un costruttore utilizzando il costrutto "class"

class NamedPerson {
   constructor(_firstName, _lastName) {
      this.firstName = _firstName
      this.lastName = _lastName
   }
   greet() {
      return "Hello, my name is " + this.firstName
   }
}

// la creazione di oggetti tramite "class" è identica
const p1 = new NamedPerson("Mario", "Bianchi")
// p1 è ora un'istanza della classe NamedPerson

console.log(p1.greet())
// "Hello, my name is Mario"
```

Una funzionalità disponibile solamente utilizzando il costrutto “class” tuttavia c’è: si tratta della possibilità di creare una nuova classe estendendo una già esistente, utilizzando tramite la keyword “extends”. Utilizzando “extends” si crea una sottoclasse a partire dall’originale, che erediterà tutti gli attributi/metodi di quella iniziale permettendo di aggiungerne di nuovi. E possibile dunque invocare dalla classe estesa anche i metodi/attributi della classe originale.

```
class Developer extends NamedPerson {
   // extends consente di dichiarare una nuova classe
   // derivata da Person
   constructor(name, surname, yearsOfExp) {
      // all'interno del costruttore come prima cosa
      // invochiamo il costruttore della classe da cui
      // estendiamo tramite la keyword "super"
      super(name, surname)
      // successivamente possiamo aggiungere nuove
      // proprietà specifiche per la classe Developer
      this.yearsOfExperience = yearsOfExp
      this.knownLanguages = []
   }
}

const dev1 = new Developer('Giuseppe', 'Verdi', 5)
console.log(dev1.yearsOfExperience) // 5
console.log(dev1.greet()) // greet() esiste su NamedPerson
```

Se definiamo una sottoclasse a partire da un’altra, molto probabilmente all’interno del suo costruttore sarà necessario come prima cosa invocare il costruttore della classe da cui stiamo estendendo. **Per farlo è sufficiente invocare “super()”. “super()”** è un riferimento alla funzione costruttore della classe originale. Nell’esempio a fianco, il costruttore della classe Developer riceve tre parametri: due saranno passati al costruttore della classe Person per la sua inizializzazione, mentre yearsOfExp viene assegnato ad un attributo presente solamente negli oggetti di classe developer

```
class Developer extends NamedPerson {
   // extends consente di dichiarare una nuova classe
   // derivata da Person
   constructor(name, surname, yearsOfExp) {
      // all'interno del costruttore come prima cosa
      // invochiamo il costruttore della classe da cui
      // estendiamo tramite la keyword "super"
      super(name, surname)
      // successivamente possiamo aggiungere nuove
      // proprietà specifiche per la classe Developer
      this.yearsOfExperience = yearsOfExp
      this.knownLanguages = []
   }
}

const dev1 = new Developer('Giuseppe', 'Verdi', 5)
console.log(dev1.yearsOfExperience) // 5
console.log(dev1.greet()) // greet() esiste su NamedPerson
// "Hello, my name is Giuseppe"
```

“super”, più genericamente, rappresenta un riferimento alla classe che stiamo estendendo. Tramite esso è possibile anche invocare nella classe estesa metodi e proprietà di quella originaria, come indicato nell’esempio a fianco. Il metodo “meows” della classe Cat altro non fa che invocare il metodo “roars” della classe Lion. **“super”,** più genericamente, rappresenta un riferimento alla classe che stiamo estendendo. Tramite esso è possibile anche invocare nella classe estesa metodi e proprietà di quella originaria, come indicato nell’esempio a fianco. Il metodo “meows” della classe Cat altro non fa che invocare il metodo “roars” della classe Lion.

La keyword "static" definisce un metodo statico o un attributo statico per una classe. I membri statici possono venire richiamati solamente sulla classe stessa, e non verranno trovati all’interno delle istanze generate a partire dalla classe. I metodi statici vengono spesso usati per creare funzioni di utilità, spesso atte alla creazione/clonazione di oggetti, mentre le proprietà statiche sono utili per cache, configurazione o per la definizione di qualsiasi altro dato non sia necessario replicare nelle istanze.

```
// definisco una classe Point
class Point {
   // scrivo un costruttore per inizializzare due proprietà
   constructor(x, y) {
      this.x = x
      this.y = y
   }
   // aggiungo un attributo statico "displayName"
   static displayName = "Point"
}

// creo p1 come istanza della classe Point
const p1 = new Point(5, 5)

// cerco di leggere il valore di "displayName" su p1
p1.displayName  // undefined

// ne leggo correttamente il valore se lo cerco sulla classe
console.log(Point.displayName)  // "Point"
```

Lakeyword extendspermette di “estendere” una classe all’interno di un’altra = si crea una sottoclasse a partire dall’originale, che erediterà tutti gli attributi/metodi di quella iniziale permettendo di aggiungerne di nuovi.

In questo caso la nuova classe Developer avrà al tuo interno tutte le proprietà della classe Person.

La keyword SUPER= invoca il costruttore della super-classe da cui Developer si estende, vale a dire Person.Person è quindi definita laSUPERCLASSEdi Developer.

```
ùconst formNode = document.querySelector("form");
const nome = document.getElementById("fname");
const proprietario = document.getElementById("lname");
const specie = document.getElementById("species");
const razza = document.getElementById("breed");

const animalss = [];

class animal {
  constructor(nome, proprietario, specie, razza) {
    this.nome = nome;
    this.proprietario = proprietario;
    this.specie = specie;
    this.razza = razza;
  }
}
formNode.onsubmit = function (e) {
  e.preventDefault();
  const p = document.createElement("p");
  const div = document.querySelector("div");
  div.appendChild(p);
  const animali = new animal(nome.value, proprietario.value, specie.value, razza.value);
  animalss.push(animali);

  p.textContent =
    "L'animale è " +
    animali.nome +
    "  " +
    "Il proprietario è" +
    animali.proprietario +
    "  " +
    "La specie è" +
    animali.specie +
    "  " +
    "La razza è" +
    animali.razza;

  nome.value = "";
  proprietario.value = "";
  specie.value = "";
  razza.value = "";
};
```