wantsome
The friendly IT Academy

Saptamana 12

Partea 1

Programare Front-End

2. OOP


OBJECT
INHERITANCE
CLASS
1
2
3
ENCAPSULATION

OOP – **O**bject **O**riented **P**rogramming

Programarea Orientată Obiect (POO) este o metoda de proiectare si implementare in care programele sunt reprezentate sub forma unor colectii de obiecte (clase) care interactioneaza intre ele prin intermediul unor mesaje

Caracteristicile ale OOP-ului:

- *Clase*
- *Obiecte*
- *Metode*
- *Proprietati*

2.1 Principii si concepte de baza in OOP

Concepte de baza in OOP

Obiect - reprezentarea a unei entitati din lumea reala asupra caruia se pot intreprinde una sau mai multe actiuni si care la randul sau poate intreprinde actiuni

Un obiect este caracterizat de:

- **Nume**
- **Atribute** (date a caror valoare la un moment dat reprezinta starea obiectului)
- **Metode** (operatii / servicii care se executa asupra atributelor sau folosind valoarea lor)

Concepte de baza in OOP

Clasa - o colectie de obiecte care impartasesc aceeasi ***structura*** (caracteristici) si acelasi ***comportament*** (metode sau operatii)

Clasa Bicicleta

- atribute

tip cadru

dimensiune roata

numar viteze

- metode

accelereaza

franeaza

Obiect: DHS

- atribute

mountain

21

21

- metode

accelereaza

franeaza

Obiect: VanMoof

- atribute

city

21

3

- metode

accelereaza

franeaza

Identificarea si stabilirea atributelor si metodelor

Exemplu:

- In cadrul unei banci, un cont bancar are: titular, sold, o rata a dobanzii, IBAN
- Folosind un cont se pot efectua operatii precum: depunere, extragere, interogare sold

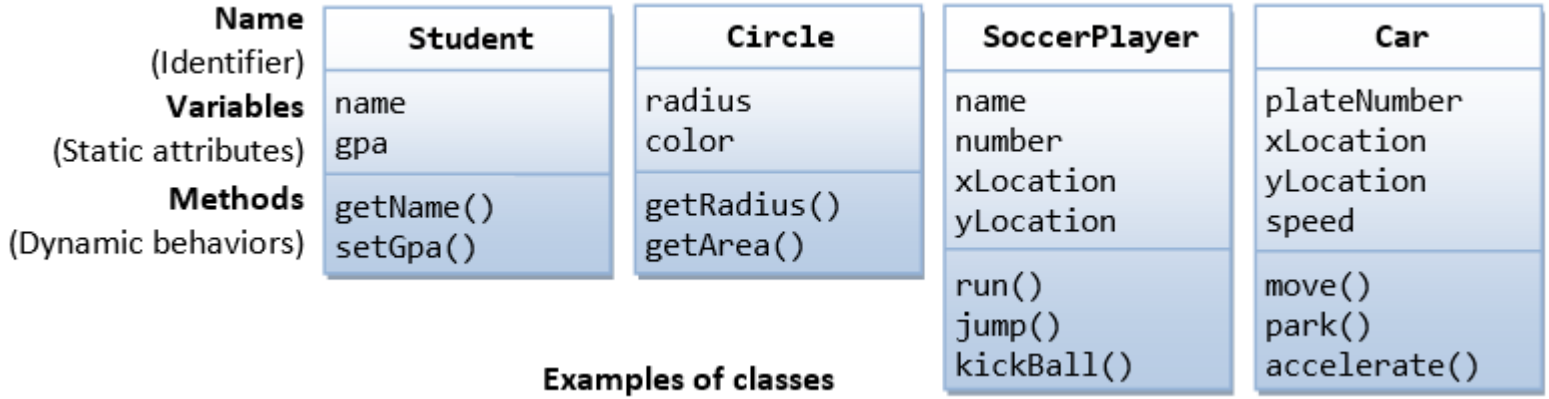
Atribute: titular, sold, rata dobanzii, IBAN

Metode: depunere, extragere, interogare

De retinut

- o clasa este o implementare a unui tip de date abstract, definind atributele si metodele care implementeaza structura de date, respectiv operatiile tipului de date abstract

- un obiect este o instanta a unei clase fiind unic determinat de numele sau si are o stare reprezentata de valorile atributelor sale la un moment dat



Examples of classes

Principii de baza in OOP

Abstractizare

Incapsulare

Modularizare

Ierarhizare

Abstractizarea

- reprezinta procesul de grupare a datelor si metodelor de prelucrare specifice rezolvarii unei probleme

    - exprima toate caracteristicile esentiale ale unui obiect care il fac pe acesta sa se distinga de alte obiecte

    - conceptual, ofera o perspectiva clara a granitelor obiectelor din perspectiva unui privitor extern

Incapsularea

- ascunde detaliile implementarii unui obiect
- se refera la gruparea datelor si metodelor care sunt aplicabile pe acestea intr-o singura structura de date, definind totodata modul in care obiectul in sine sau programul le pot accesa
- defineste apartenenta unor proprietati si metode fata de un obiect
- separa aspectele externe ale unui obiect care sunt accesibile altor obiecte de aspectele interne ale obiectului care sunt ascunse celorlalte obiecte

Modularizarea

- modalitatea prin care un program este divizat in subunitati (module)
- un modul grupeaza abstractiuni (clase) care sunt legate din punct de vedere logic intre ele

Ierarhizarea

- ordonare a abstractiunilor prin:

  Mostenire (ierarhie de clase) - relatie intre clase in care o clasa preia structura si comportamentul (proprietatile si metodele) definit in una sau mai multe clase (*o clasa este …)*

  Agregare (ierarhie de obiecte) - relatie intre doua obiecte in care unul dintre acestea apartine celuilalt (*un obiect face parte din …*)

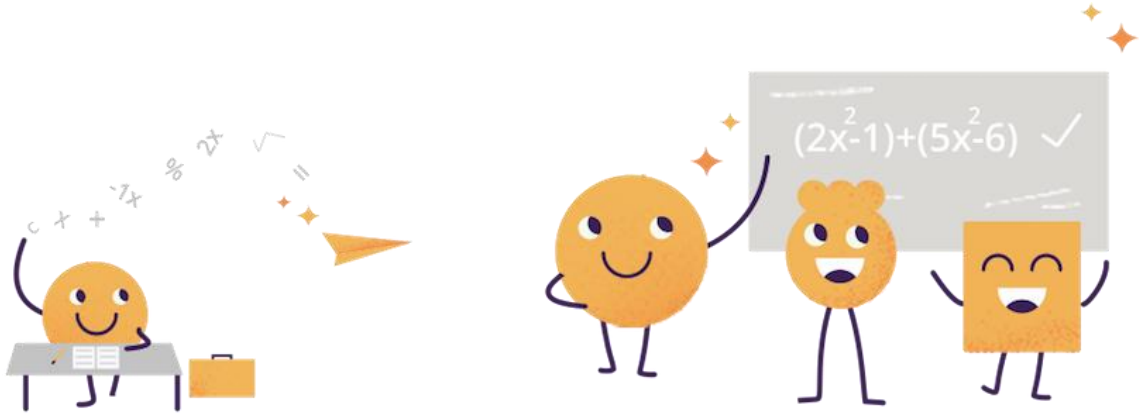
3. OOP in JavaScript

3.1 JavaScript Object Constructors

JavaScript Object Constructors

https://www.w3schools.com/js/js_object_constructors.asp

PRACTICE: **JavaScript Object Constructors**



3.2 Prototypes

What is prototype?

Notiunea de **prototype** in JS are la baza doua chestiuni importante:

1. Fiecare functie JavaScript are o proprietate **prototype** (empty by default) pe care putem atasa metode si proprietati (pentru ca este un obiect).
   O gasim sub denumirea de __proto__ (pseudo-proprietate) in cele mai noi browsere.

```
protoObject = {
  name: "test",
  number: 2
}
console.log(protoObject)
```

```
{name: "test", number: 2}
  name: "test"
  number: 2
  __proto__:
    constructor: ƒ Object()
    hasOwnProperty: ƒ hasOwnProperty()
    isPrototypeOf: ƒ isPrototypeOf()
    propertyIsEnumerable: ƒ propertyIsEnumerable()
    toLocaleString: ƒ toLocaleString()
    toString: ƒ toString()
    valueOf: ƒ valueOf()
```

Example: *How to access prototype:* **http://bit.do/protoExample**

What is prototype?

2. **prototype attribute**, definita ca si proprietate a unui obiect - aceasta specifica obiectul *parinte* (de la cine s-au mostenit metodele si proprietatile existente)

- Toate obiectele mostenesc (inherit) proprietati si metode de la un **prototype**

What is prototype?

Toate obiectele care **mostenesc** de la alte obiecte, mostenesc si proprietatea ***constructor***, care face referinta la ***constructor*-ul** obiectului.

```
function Account () {
}
var myObj = new Object ();
// And if you later want to find the myObj constructor
console.log(myObj.constructor); // Object()

// Another example: Account () is the constructor
var userAccount = new Account ();
// Find the userAccount object's constructor
console.log(userAccount.constructor); // Account()
```

Why is prototype important ? When to use it ?

1. **Prototype Property: Prototype-based Inheritance**
   - **JavaScript,** spre deosebire de alte limbaje, nu poate face mostenirea dupa clase, la fel ca in limbaje precum C# sau Java, precum ca aceste concept nu exista in definitia de baza a limbajului
   - Din acest motiv, paradigma OOP in JavaScript este bazata pe prototipuri la nivel de obiecte

Inheritance - mostenenirea de metode si proprietati de la alte obiecte

In JavaScript implementam *inheritance-ul (mostenirea)* cu ajutorul **prototype**

Example of inheritance : **http://bit.do/inheritanceExample**

Why is prototype important ? When to use it ?

2. **Prototype Attribute: Accessing Properties on Objects**
   Prototype attribute sau *prototype object* este obiectul parinte “de la care s-a mostenit”
   - Analogie: mostenirea numelui de familie al tatalui

- Atunci cand obiectele se creeaza, se cauta acest *prototype object* pana se gaseste

- Daca exista o succesiune de obiecte create, se merge pe “scara” pana la ultimul obiect (bunicul)

- De aici si denumirea de **prototype chain**

Prototype chain

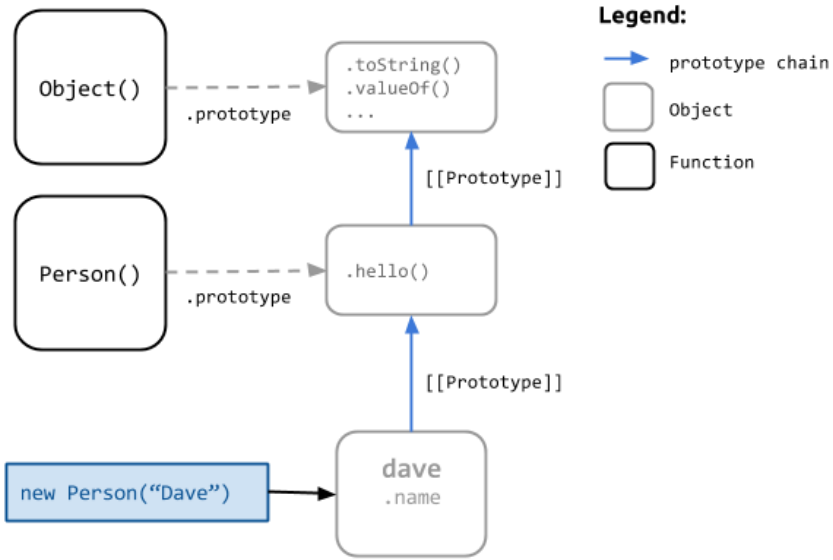
Prototype chain - Inlantuirea din object prototype in object prototype (nested).

- JS foloseste acest **chain** pentru a “gasi” proprietati si metode definite

- Cautarea unei anumite proprietati sau metode si negasirea acesteia in lantul de *prototypes* va returna `undefined`

Example : **http://bit.do/chainExample**

Prototype chain



Proto1 proto 2 explained

PRACTICE: **Prototype and inheritance**

http://bit.do/exProto
http://bit.do/ex1proto
http://bit.do/exProtoBonus