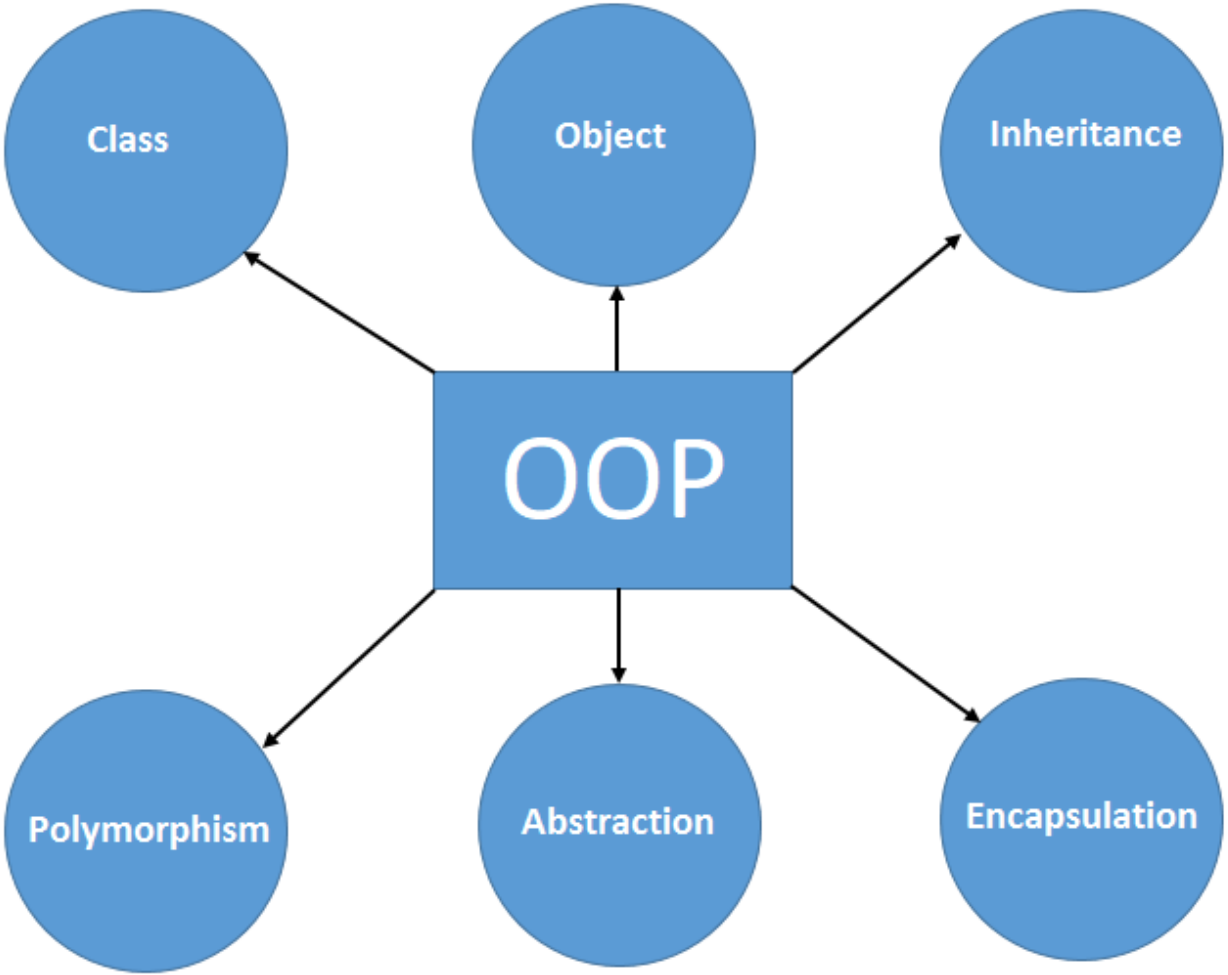
4. Advanced JavaScript Programming

Adrian Adiaconitei

LINK Academy

Obiective

- ✓ Recapitulare
- ✓ Tratarea erorilor
- ✓ Design patterns



JavaScript – OOP: class / object

```
class ClassName {
    constructor() {
        // constructor logic
    }
    method1() {
        // method1 logic
    }
    method2() {
        // method2 logic
    }
}
```

Ap1.html

```
const myObj  = new ClassName();
const myObj2 = new ClassName();
```

JavaScript – OOP: Inheritance

```
class ChildClass extends ParentClass {
        constructor() {
                super();
                // child constructor logic
        }
        method1() {
                // child method1 logic
        }
        method2() {
                super.method2();
                // call parent method2 // child method2 logic
        }
}
```

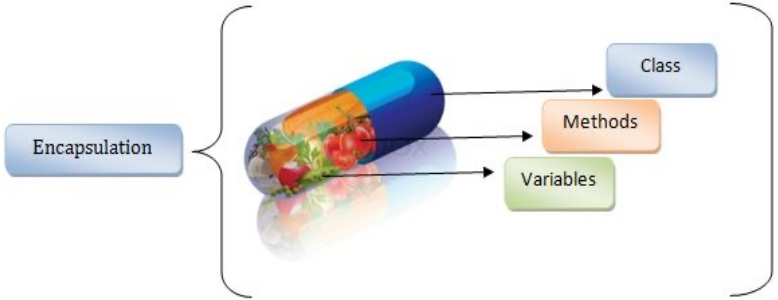
Ap2.html

JavaScript – OOP: Encapsulation

```
class Website {
	name; // public field
	constructor(name) {
		this.name = name;
	}
	// public method
	greet() {
		console.log(`Hello, welcome to ${this.name}.`);
	}
}
let obj = new Website('https://link-academy.com/');
obj.greet()
```

Ap3.html

- ✓ **Public**
- ✓ **Private**
- ✓ **Protected**
- ✓ **Getters & Setters**
  - ✓ get
  - ✓ set



JavaScript – OOP: Encapsulation

În JavaScript, există **două** tipuri de membrii obiect (proprietăți și metode):

- ✓ **public** : accesibili de oriunde, din interior și din afara clasei.
- ✓ **private** (#) : accesibili numai din interiorul clasei. Metode și proprietăți, accesibile din alte metode ale clasei, dar nu din exterior și nu pot fi moșteniți

- ✓ protected (_) : **JavaScript nu are suport nativ pentru membrii protejați**, care sunt accesibile în cadrul clasei și subclaselor sale, dar nu în afară. Este o modalitate de a semnala altor dezvoltatori că ar trebui să respecte principiul încapsulării și să nu acceseze sau să modifice acești membri direct. Este important de reținut că convenția nu împiedică pe nimeni să acceseze sau să modifice acești membri dacă își dorește cu adevărat. **Este doar o chestiune de bună practică și de lizibilitate a codului.**

JavaScript – OOP: Abstraction

```
class Person {
      constructor() {
         if (this.constructor == Person) {   throw new Error("Your Error Message...");   }
      }
      info() {
          throw new Error(" Added abstract Method has no implementation");
      }
}
class Teacher extends Person {
   info() {    console.log("I am a Teacher");    }
}
var teacher1 = new Teacher();
teacher1.info();
```

Ap4.html

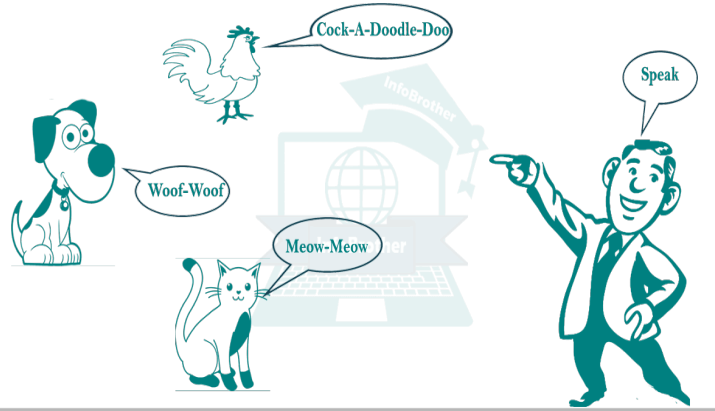
JavaScript – OOP: Abstraction

- ✓ Abstractizarea este o modalitate de a crea un model simplu al unor entități mai complexe din lumea reală. Abstractizarea permite ca entitățile complexe din lumea reală să fie modelate la nivel de limbaj de programare.
- ✓ JavaScript nu suportă în mod nativ conceptul de **clasă abstractă**, iar clasa abstractă este o clasă care poate fi doar extinsă și nu instanțiată. O **clasă abstractă** este o clasă care conține una sau mai multe metode abstracte, dar **nu poate instanția.**

JavaScript – OOP: Polymorphism

```
class A {
      display() {  document.writeln("A is invoked<br>");   }
}
class B extends A {
      display() {   document.writeln("B is invoked")   }
}
let a = [new A(), new B()]
a.forEach(function (msg) {
    msg.display();
});
```

Ap5.html



JavaScript - OOP

Aplicatia 1:Listă cărți

- ține evidenta cărților: titlu, autor, ISBN
- adaugă / șterge carte
- informațiile se vor păstra în Local Storage
- se recomandă să folosim module

JavaScript - OOP

Aplicatia 2:To Do List

- ține evidenta sarcinilor titlu, descriere, data
- adaugă / editează / șterge sarcină
- informațiile se vor păstra în Local Storage
- se recomandă să folosim module

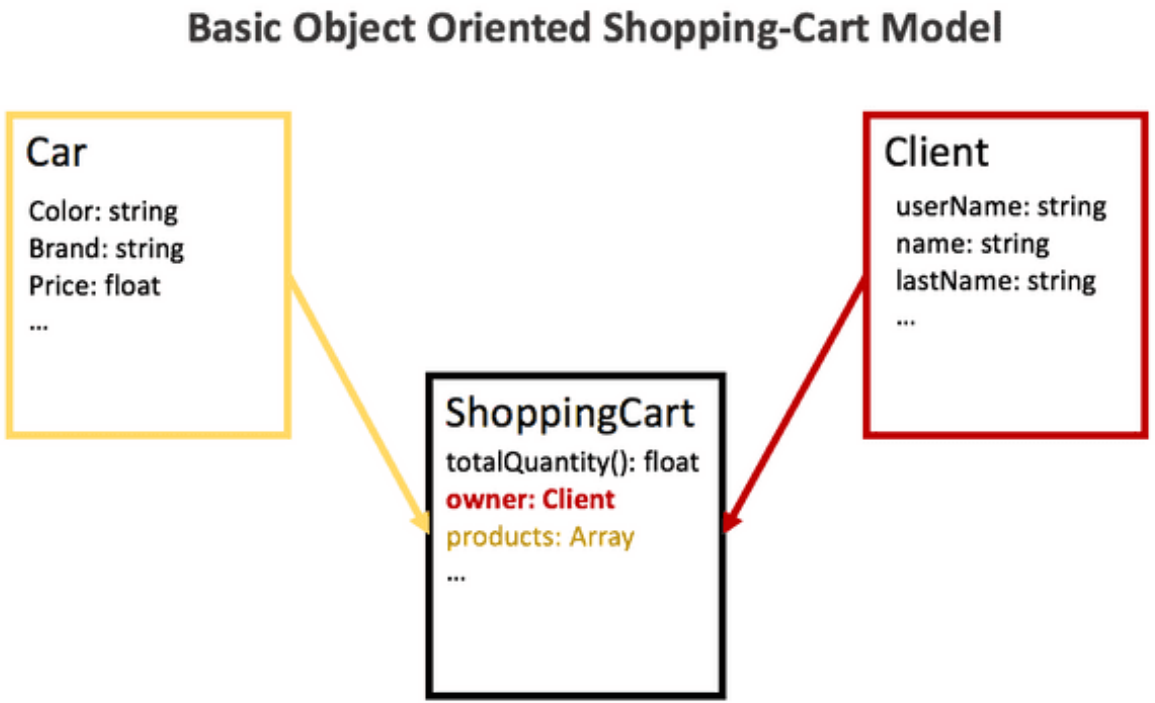
JavaScript - OOP

Provocare:Site web de dealer auto

- ține evidenta mașinilor: nume,model, culoare,pret.... descriere
- adaugă / editează / șterge mașină
- informațiile se vor păstra în Local Storage
- se recomandă să folosim module

JavaScript - OOP

Provocare:Site web de dealer auto



JavaScript – OOP

Joc 1: Tetris
https://github.com/certif/Tetris

Joc 2: Găsește fraza corectă
https://github.com/Marcelckp/oop_Game_Show_App-v2

Provocare: folosiți module

JavaScript – Tratarea erorilor

- ✓ **Javascript Promises**

  .then(), .catch(), .finally()

- Website Building – curs 5

JavaScript – Tratarea erorilor

try un bloc de cod cd trebuie executat/rulat

catch un bloc de cod pentru a gestiona orice eroare.

finally un bloc de cod de rulat indiferent de rezultat.

throw definește o eroare personalizată.

JavaScript – Tratarea erorilor

```
try {
        // code that we will 'try' to run
        // throw
} catch(error) {
        // code to run if there are any problems
}
```

```
try {
        // code that we will 'try' to run
        // throw
} catch(error) {
        // code to run if there are any problems
} finally {
        // run this code no matter what the previous outcomes
}
```

JavaScript – Tratarea erorilor

https://www.w3schools.com/js/tryit.asp?filename=tryjs_finally_error

Ap6.html

Ap7.html

Ap8.html

JavaScript – Design patterns

- ✓ Design patterns - soluții reutilizabile la problemele frecvente în proiectarea software.

- ✓ Design patterns - ne oferă, de asemenea, un vocabular comun pentru a descrie soluțiile.

JavaScript – Design patterns

- ✓ **Factory -** este un obiect care creează alte obiecte.
- ✓ Dacă aplicația dvs. are nevoie de mai mult control asupra procesului de creare a obiectelor, luați în considerare utilizarea unei fabrici.

Ap11.html

Ap12.html

JavaScript – Design patterns

- ✓ **Singleton**

- ✓ Ap11.html

Resurse

- ✓ https://github.com/FaztWeb/javascript-products-app-oop
- ✓ https://www.frontendmentor.io/challenges/tip-calculator-app-ugJNGbJUX
- ✓ https://github.com/yasssuz/tip-calculator-app-oop
- ✓ https://www.patterns.dev/posts/classic-design-patterns/
- ✓ https://github.com/sag1v/OOP-Javascript-Presentation
- ✓ https://sag1v.github.io/OOP-Javascript-Presentation/#/
- ✓ https://www.dofactory.com/javascript/design-patterns
- ✓ https://www.patterns.dev/posts/classic-design-patterns/