**Atelier 2 : Javascript et POO**

**Objectif :**L’objectif principal de ce Travail pratique et de se familiariser avec les conceptes POO de langage javascript.

**Exercice 1 :**

1. Ecrire une fonction constructeur voiture avec les attributs « model , marque, année, type, carburant » .

function Voiture(model, marque, annee, type, carburant) {

this.model = model;

this.marque = marque;

this.annee = annee;

this.type = type;

this.carburant = carburant;

}SSSS

L'exécution

[Execution complete with exit code 0]

2. créer une liste des voitures.

let voitures = [

new Voiture("Model 3", "Tesla", 2022, "Berline", "Électrique"),

new Voiture("i30", "Hyundai", 2019, "Compacte", "Essence"),

new Voiture("Focus", "Ford", 2021, "SUV", "Diesel")

];

// Affichage

console.log(voitures);

L'exécution

[

Voiture {

model: 'Model 3',

marque: 'Tesla',

annee: 2022,

type: 'Berline',

carburant: 'Électrique'

},

Voiture {

model: 'i30',

marque: 'Hyundai',

annee: 2019,

type: 'Compacte',

carburant: 'Essence'

},

Voiture {

model: 'Focus',

marque: 'Ford',

annee: 2021,

type: 'SUV',

carburant: 'Diesel'

}

]

3. implémenter le mécanisme d’héritage entre la classes voitures et deux autre classes Hyndai ( attributs : série(string) , hybride (booléen) / méthodes : alarmer() )et Ford ( options (tableau )).

function Hyundai(model, marque, annee, type, carburant, serie, hybride) {

Voiture.call(this, model, marque, annee, type, carburant); // Appel du constructeur parent

this.serie = serie;

this.hybride = hybride;

this.alarmer = function () {

console.log("Alerte Hyundai !");

};

}

Hyundai.prototype = Object.create(Voiture.prototype);

Hyundai.prototype.constructor = Hyundai;

function Ford(model, marque, annee, type, carburant, options) {

Voiture.call(this, model, marque, annee, type, carburant); // Appel du constructeur parent

this.options = options;

}

Ford.prototype = Object.create(Voiture.prototype);

Ford.prototype.constructor = Ford;

L'exécution

[

Voiture {

model: 'Clio',

marque: 'Renault',

annee: 2018,

type: 'Citadine',

carburant: 'Essence'

},

Hyundai {

model: 'Tucson',

marque: 'Hyundai',

annee: 2020,

type: 'SUV',

carburant: 'Hybride',

serie: 'H1',

hybride: true,

alarmer: [Function (anonymous)]

},

Ford {

model: 'Kuga',

marque: 'Ford',

annee: 2021,

type: 'SUV',

carburant: 'Diesel',

options: [ 'GPS', 'Caméra', 'Toit ouvrant' ]

}

]

4. Trier puis afficher les voiture selon un ordre croissant des année.

// Création de la liste complète de voitures

let voitures = [

new Voiture("Clio", "Renault", 2018, "Citadine", "Essence"),

new Hyundai("Tucson", "Hyundai", 2020, "SUV", "Hybride", "H1", true),

new Ford("Kuga", "Ford", 2021, "SUV", "Diesel", ["GPS", "Caméra", "Toit ouvrant"])

];

// Tri croissant selon l’année

voitures.sort((a, b) => a.annee - b.annee);

// Affichage

console.log(voitures);

L'exécution

[

Voiture {

model: 'Clio',

marque: 'Renault',

annee: 2018,

type: 'Citadine',

carburant: 'Essence'

},

Hyundai {

model: 'Tucson',

marque: 'Hyundai',

annee: 2020,

type: 'SUV',

carburant: 'Hybride',

serie: 'H1',

hybride: true,

alarmer: [Function (anonymous)]

},

Ford {

model: 'Kuga',

marque: 'Ford',

annee: 2021,

type: 'SUV',

carburant: 'Diesel',

options: [ 'GPS', 'Caméra', 'Toit ouvrant' ]

}

]

**Exercice 2 :**

1. Creer deux objet native Etudiant ( nom : string , prenom : string , age : number , cne : string ) et Professeur ( nom : string , age : number , cin : string , )

// Objet Etudiant

let etudiant = {

nom: "El Amrani",

prenom: "Nadia",

age: 21,

cne: "CNE123456"

};

// Objet Professeur

let professeur = {

nom: "Mr. Karim",

age: 45,

cin: "CIN789012"

};

L'exécution

[Execution complete with exit code 0]

2. Ajouter une méthode étudier() a l’objet Etudiant, puis ajouter une méthode enseigner() a l’objet Etudiant Professeur.

// Ajout de la méthode 'etudier'

etudiant.etudier = function() {

console.log(this.prenom + " " + this.nom + " est en train d'étudier.");

};

etudiant.etudier();

L'exécution

Nadia El Amrani est en train d'étudier.

// Ajout de la méthode 'enseigner'

professeur.enseigner = function () {

console.log(this.nom + " " + this.nom + " est en train d'enseigner.");

};

professeur.enseigner();

L'exécution

Mr. Karim est en train d'enseigner.

3. Trier les étudiants selon l’ordre alphabétique « nom , prénom et age».

// Liste d'étudiants

let etudiants = [

{ nom: "Zahraoui", prenom: "Omar", age: 22 },

{ nom: "Alaoui", prenom: "Salma", age: 20 },

{ nom: "El Amrani", prenom: "Nadia", age: 21 },

{ nom: "Zahraoui", prenom: "Imane", age: 22 }

];

// Tri alphabétique

etudiants.sort((a, b) => {

return a.nom.localeCompare(b.nom) ||

a.prenom.localeCompare(b.prenom) ||

a.age - b.age;

});

// Affichage

console.log(etudiants);

L'exécution

[

{ nom: 'Alaoui', prenom: 'Salma', age: 20 },

{ nom: 'El Amrani', prenom: 'Nadia', age: 21 },

{ nom: 'Zahraoui', prenom: 'Imane', age: 22 },

{ nom: 'Zahraoui', prenom: 'Omar', age: 22 }

]

**Exercice 3 :**

// Vecteur2D

class Vecteur2D {

constructor(x = 0, y = 0) {

this.x = x;

this.y = y;

}

afficher() {

return (${this.x}, ${this.y});

}

addition(v) {

return new Vecteur2D(this.x + v.x, this.y + v.y);

}

}

let v1 = new Vecteur2D();

let v2 = new Vecteur2D(3, 4);

console.log(v1.afficher(), v2.afficher(), v1.addition(v2).afficher());

// Rectangle et Carré

class Rectangle {

constructor(longueur = 1, largeur = 1) {

this.nom = "rectangle";

this.longueur = longueur;

this.largeur = largeur;

}

afficher() {

return ${this.nom}: ${this.longueur} x ${this.largeur};

}

surface() {

return this.longueur \* this.largeur;

}

}

class Carre extends Rectangle {

constructor(cote = 1) {

super(cote, cote);

this.nom = "carré";

}

}

let r = new Rectangle(5, 3);

let c = new Carre(4);

console.log(r.afficher(), "Surface:", r.surface());

console.log(c.afficher(), "Surface:", c.surface());

// Point et Segment

class Point {

constructor(x = 1.1, y = 4.5) {

this.x = x;

this.y = y;

}

}

class Segment {

constructor(x1, y1, x2, y2) {

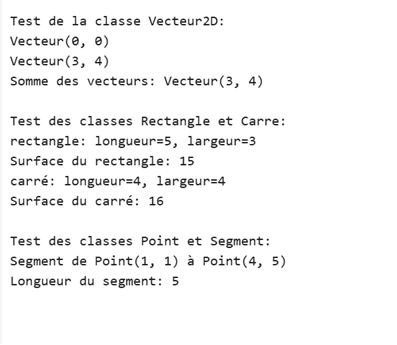
this.orig = new Point(x1, y1);

this.extrem = new Point(x2, y2);

}

}

L'exécution



**Exercice 4 :**

]class User {

constructor(id, username, email) {

this.id = id;

this.username = username;

this.email = email;

} toJSON() {

return {

id: this.id,

username: this.username,

email: this.email

};

}

static fromJSON(json) {

return new User(json.id, json.username, json.email);

}

}

class Post {

constructor(id, title, content, userId, date = new Date()) {

this.id = id;

this.title = title;

this.content = content;

this.userId = userId;

this.date = date;

}

toJSON() {

return {

id: this.id,

title: this.title,

content: this.content,

userId: this.userId,

date: this.date.toISOString()

};

}

static fromJSON(json) {

return new Post(

json.id,

json.title,

json.content,

json.userId,

new Date(json.date)

);

}

}

class Blog {

constructor() {

this.users = [];

this.posts = [];

this.nextUserId = 1;

this.nextPostId = 1;

}

addUser(username, email) {

const user = new User(this.nextUserId++, username, email);

this.users.push(user);

return user;

}

addPost(title, content, userId) {

const post = new Post(this.nextPostId++, title, content, userId);

this.posts.push(post);

return post;

}

getAllPosts() {

return this.posts;

}

getPostsByUser(userId) {

return this.posts.filter(post => post.userId === userId);

}

toJSON() {

return {

users: this.users.map(user => user.toJSON()),

posts: this.posts.map(post => post.toJSON()),

nextUserId: this.nextUserId,

nextPostId: this.nextPostId

};

}

static fromJSON(json) {

const blog = new Blog();

blog.nextUserId = json.nextUserId;

blog.nextPostId = json.nextPostId;

blog.users = json.users.map(User.fromJSON);

blog.posts = json.posts.map(Post.fromJSON);

return blog;

}

saveToFile(fs, filename) {

const json = JSON.stringify(this.toJSON(), null, 2);

fs.writeFileSync(filename, json);

console.log(`Blog sauvegardé dans ${filename}`);

}

static loadFromFile(fs, filename) {

try {

const json = JSON.parse(fs.readFileSync(filename, 'utf8'));

return Blog.fromJSON(json);

} catch (error) {

console.error(`Erreur lors du chargement depuis ${filename}:`, error.message);

return new Blog(); // Retourner un nouveau blog vide en cas d'erreur

}

}

}

const blog = new Blog();

const user1 = blog.addUser("john\_doe", "john@example.com");

const user2 = blog.addUser("jane\_smith", "jane@example.com");

blog.addPost("Premier article", "Contenu du premier article", user1.id);

blog.addPost("Deuxième article", "Contenu du deuxième article", user1.id);

blog.addPost("Article de Jane", "Contenu de l'article de Jane", user2.id);

console.log("Tous les posts:");

blog.getAllPosts().forEach(post => {

console.log(`${post.title} (par utilisateur #${post.userId})`);

});

console.log("\nPosts de l'utilisateur #1:");

blog.getPostsByUser(user1.id).forEach(post => {

console.log(`${post.title}: ${post.content}`);

});

console.log("\nDémonstration de conversion en JSON:");

const blogJSON = JSON.stringify(blog.toJSON(), null, 2);

console.log(blogJSON);

const fs = require('fs');

blog.saveToFile(fs, 'blog-ata.json');

const loadedBlog = Blog.loadFromFile(fs, 'blog-data.json');

console.log("\nBlog chargé depuis le fichier:");

console.log(`Nombre d'utilisateurs: ${loadedBlog.users.length}`);

console.log(`Nombre de posts: ${loadedBlog.posts.length}`);

L'exécution

Tous les posts:

Premier article (par utilisateur #1)

Deuxième article (par utilisateur #1)

Article de Jane (par utilisateur #2)

Posts de l'utilisateur #1:

Premier article: Contenu du premier article

Deuxième article: Contenu du deuxième article

Démonstration de conversion en JSON:

{

"users": [

{

"id": 1,

"username": "john\_doe",

"email": "john@example.com"

},

{

"id": 2,

"username": "jane\_smith",

"email": "jane@example.com"

}

],

"posts": [

{

"id": 1,

"title": "Premier article",

"content": "Contenu du premier article",

"userId": 1,

"date": "2025-04-24T21:09:30.896Z"

},

{

"id": 2,

"title": "Deuxième article",

"content": "Contenu du deuxième article",

"userId": 1,

"date": "2025-04-24T21:09:30.896Z"

},

{

"id": 3,

"title": "Article de Jane",

"content": "Contenu de l'article de Jane",

"userId": 2,

"date": "2025-04-24T21:09:30.896Z"

}

],

"nextUserId": 3,

"nextPostId": 4

}

Blog sauvegardé dans blog-ata.json

Erreur lors du chargement depuis blog-data.json: ENOENT: no such file or directory, open 'blog-data.json'

Blog chargé depuis le fichier:

Nombre d'utilisateurs: 0

Nombre de posts: 0

**Exercice 5 :**

console.log("1. GESTION DES LIVRES (ARRAY):");

let books = ["Le Petit Prince", "1984", "Harry Potter"];

console.log("Tableau initial:", books);

books.push("Don Quichotte");

console.log("Après push:", books);

books.unshift("Les Misérables");

console.log("Après unshift:", books);

let removedLastBook = books.pop();

console.log("Livre retiré avec pop:", removedLastBook);

console.log("Après pop:", books);

let removedFirstBook = books.shift();

console.log("Livre retiré avec shift:", removedFirstBook);

console.log("Après shift:", books);

console.log("\n2. GESTION DES CATÉGORIES (SET):");

let categories = new Set(["Fiction", "Science", "Histoire"]);

console.log("Set initial:", categories);

categories.add("Philosophie");

console.log("Après ajout de 'Philosophie':", categories);

categories.add("Fiction");

console.log("Après tentative d'ajout de 'Fiction' (déjà existant):", categories);

categories.delete("Science");

console.log("Après suppression de 'Science':", categories);

console.log("\n3. GESTION DES EMPRUNTS (MAP):");

let borrows = new Map();

borrows.set("Le Petit Prince", "Alice");

borrows.set("1984", "Bob");

borrows.set("Don Quichotte", "Charlie");

console.log("Map initial:");

borrows.forEach((emprunteur, livre) => {

console.log(`- "${livre}" emprunté par ${emprunteur}`);

});

borrows.delete("1984");

console.log("\nAprès suppression de '1984':");

borrows.forEach((emprunteur, livre) => {

console.log(`- "${livre}" emprunté par ${emprunteur}`);

});

let bookToCheck = "Le Petit Prince";

console.log(`\nVérification si "${bookToCheck}" est emprunté:`, borrows.has(bookToCheck));

bookToCheck = "Harry Potter";

console.log(`Vérification si "${bookToCheck}" est emprunté:`, borrows.has(bookToCheck));

L'exécution

1. GESTION DES LIVRES (ARRAY):

Tableau initial: [ 'Le Petit Prince', '1984', 'Harry Potter' ]

Après push: [ 'Le Petit Prince', '1984', 'Harry Potter', 'Don Quichotte' ]

Après unshift: [

'Les Misérables',

'Le Petit Prince',

'1984',

'Harry Potter',

'Don Quichotte'

]

Livre retiré avec pop: Don Quichotte

Après pop: [ 'Les Misérables', 'Le Petit Prince', '1984', 'Harry Potter' ]

Livre retiré avec shift: Les Misérables

Après shift: [ 'Le Petit Prince', '1984', 'Harry Potter' ]

2. GESTION DES CATÉGORIES (SET):

Set initial: Set(3) { 'Fiction', 'Science', 'Histoire' }

Après ajout de 'Philosophie': Set(4) { 'Fiction', 'Science', 'Histoire', 'Philosophie' }

Après tentative d'ajout de 'Fiction' (déjà existant): Set(4) { 'Fiction', 'Science', 'Histoire', 'Philosophie' }

Après suppression de 'Science': Set(3) { 'Fiction', 'Histoire', 'Philosophie' }

3. GESTION DES EMPRUNTS (MAP):

Map initial:

- "Le Petit Prince" emprunté par Alice

- "1984" emprunté par Bob

- "Don Quichotte" emprunté par Charlie

Après suppression de '1984':

- "Le Petit Prince" emprunté par Alice

- "Don Quichotte" emprunté par Charlie

Vérification si "Le Petit Prince" est emprunté: true

Vérification si "Harry Potter" est emprunté: false