# **SÉANCE 2 – TypeScript, Node & Outils Modernes**

## 📚 PARTIE THÉORIQUE

### 1. Installation des outils

#### Node.js et npm

bash

*# Vérifier l'installation*

node --version

npm --version

*# Initialiser un projet*

npm init -y

*# Installer des packages*

npm install package-name

npm install --save-dev package-name *# Dev dependencies*

#### VS Code - Extensions recommandées

* ESLint
* Prettier
* Live Server
* GitLens
* Path Intellisense
* Auto Rename Tag
* JavaScript (ES6) code snippets

#### Git - Commandes de base

bash

*# Initialiser un dépôt*

git init

*# Configuration*

git config --global user.name "Votre Nom"

git config --global user.email "votre@email.com"

*# Commandes essentielles*

git add .

git commit -m "Message de commit"

git status

git log

git branch

git checkout -b nouvelle-branche

### 2. Introduction à TypeScript

#### Pourquoi TypeScript ?

* **Typage statique** : Détection d'erreurs à la compilation
* **IntelliSense amélioré** : Meilleure autocomplétion
* **Code plus maintenable** : Documentation implicite
* **Évolutivité** : Facilite les grands projets

#### Types de base

typescript

*// Types primitifs*

let nom: string = "Jean";

let age: number = 25;

let estActif: boolean = true;

let rien: null = null;

let indefini: undefined = undefined;

*// Tableaux*

let nombres: number[] = [1, 2, 3];

let textes: Array<string> = ["a", "b", "c"];

*// Tuples*

let personne: [string, number] = ["Marie", 30];

*// Enum*

enum Couleur {

Rouge = "RED",

Vert = "GREEN",

Bleu = "BLUE"

}

let maCouleur: Couleur = Couleur.Rouge;

*// Any (à éviter)*

let variable: any = "peut être n'importe quoi";

*// Unknown (préférable à any)*

let valeur: unknown = "quelque chose";

if (typeof valeur === "string") {

console.log(valeur.toUpperCase());

}

*// Void (pour fonctions sans retour)*

function afficher(): void {

console.log("Hello");

}

*// Never (pour fonctions qui ne retournent jamais)*

function erreur(message: string): never {

throw new Error(message);

}

#### Interfaces et Types

typescript

*// Interface*

interface Utilisateur {

id: number;

nom: string;

email: string;

age?: number; *// Propriété optionnelle*

readonly dateCreation: Date; *// Propriété en lecture seule*

}

*// Type Alias*

type ID = string | number;

type Point = {

x: number;

y: number;

};

*// Union Types*

type Status = "actif" | "inactif" | "suspendu";

*// Intersection Types*

type Admin = Utilisateur & {

role: "admin";

permissions: string[];

};

*// Utilisation*

const user: Utilisateur = {

id: 1,

nom: "Dupont",

email: "dupont@email.com",

dateCreation: new Date()

};

#### Fonctions typées

typescript

*// Fonction avec types*

function addition(a: number, b: number): number {

return a + b;

}

*// Arrow function*

const multiplication = (a: number, b: number): number => a \* b;

*// Fonction avec paramètres optionnels*

function saluer(nom: string, titre?: string): string {

return titre ? `Bonjour ${titre} ${nom}` : `Bonjour ${nom}`;

}

*// Fonction avec paramètres par défaut*

function creerUtilisateur(nom: string, role: string = "user"): Utilisateur {

return {

id: Math.random(),

nom,

email: `${nom}@email.com`,

dateCreation: new Date()

};

}

*// Rest parameters*

function somme(...nombres: number[]): number {

return nombres.reduce((acc, n) => acc + n, 0);

}

*// Type de fonction*

type OperationMath = (a: number, b: number) => number;

const diviser: OperationMath = (a, b) => a / b;

#### Classes en TypeScript

typescript

class Produit {

*// Propriétés*

private id: number;

public nom: string;

protected prix: number;

readonly categorie: string;

*// Constructeur*

constructor(nom: string, prix: number, categorie: string) {

this.id = Math.random();

this.nom = nom;

this.prix = prix;

this.categorie = categorie;

}

*// Méthodes*

getPrix(): number {

return this.prix;

}

setPrix(nouveauPrix: number): void {

if (nouveauPrix > 0) {

this.prix = nouveauPrix;

}

}

*// Getter*

get description(): string {

return `${this.nom} - ${this.prix}€`;

}

*// Setter*

set nouveauPrix(prix: number) {

this.setPrix(prix);

}

*// Méthode statique*

static comparer(p1: Produit, p2: Produit): number {

return p1.prix - p2.prix;

}

}

*// Héritage*

class ProduitNumerique extends Produit {

private tailleFichier: number;

constructor(nom: string, prix: number, tailleFichier: number) {

super(nom, prix, "Numérique");

this.tailleFichier = tailleFichier;

}

telecharger(): void {

console.log(`Téléchargement de ${this.nom}...`);

}

}

*// Classe abstraite*

abstract class Animal {

abstract faireUnBruit(): void;

bouger(): void {

console.log("L'animal bouge");

}

}

class Chien extends Animal {

faireUnBruit(): void {

console.log("Woof!");

}

}

#### Generics

typescript

*// Fonction générique*

function premiers<T>(tableau: T[], n: number): T[] {

return tableau.slice(0, n);

}

const nombres = premiers<number>([1, 2, 3, 4, 5], 3);

const textes = premiers<string>(["a", "b", "c"], 2);

*// Interface générique*

interface Reponse<T> {

data: T;

status: number;

message: string;

}

const reponseUtilisateur: Reponse<Utilisateur> = {

data: {

id: 1,

nom: "Dupont",

email: "dupont@email.com",

dateCreation: new Date()

},

status: 200,

message: "Succès"

};

*// Classe générique*

class Collection<T> {

private items: T[] = [];

ajouter(item: T): void {

this.items.push(item);

}

obtenir(index: number): T | undefined {

return this.items[index];

}

tous(): T[] {

return this.items;

}

}

const nombres = new Collection<number>();

nombres.ajouter(1);

nombres.ajouter(2);

### 3. Gestion des dépendances avec npm

#### package.json

json

{

"name": "mon-projet",

"version": "1.0.0",

"description": "Description du projet",

"main": "index.js",

"scripts": {

"start": "node index.js",

"dev": "nodemon index.js",

"build": "tsc",

"test": "jest"

},

"dependencies": {

"express": "^4.18.0"

},

"devDependencies": {

"typescript": "^5.0.0",

"@types/node": "^20.0.0",

"nodemon": "^3.0.0"

}

}

#### tsconfig.json

json

{

"compilerOptions": {

"target": "ES2020",

"module": "commonjs",

"lib": ["ES2020", "DOM"],

"outDir": "./dist",

"rootDir": "./src",

"strict": true,

"esModuleInterop": true,

"skipLibCheck": true,

"forceConsistentCasingInFileNames": true,

"resolveJsonModule": true,

"declaration": true,

"declarationMap": true,

"sourceMap": true

},

"include": ["src/\*\*/\*"],

"exclude": ["node\_modules", "dist"]

}

## 💻 TD 2 : EXERCICES GUIDÉS

### TD2.1 : Types de base

typescript

*// Exercice 1 : Créer des interfaces*

interface Livre {

titre: string;

auteur: string;

annee: number;

isbn?: string;

genres: string[];

}

interface Bibliotheque {

nom: string;

adresse: string;

livres: Livre[];

ajouterLivre(livre: Livre): void;

rechercherParAuteur(auteur: string): Livre[];

}

*// Exercice 2 : Implémenter la bibliothèque*

class MaBibliotheque implements Bibliotheque {

nom: string;

adresse: string;

livres: Livre[] = [];

constructor(nom: string, adresse: string) {

this.nom = nom;

this.adresse = adresse;

}

ajouterLivre(livre: Livre): void {

this.livres.push(livre);

console.log(`Livre "${livre.titre}" ajouté`);

}

rechercherParAuteur(auteur: string): Livre[] {

return this.livres.filter(livre =>

livre.auteur.toLowerCase().includes(auteur.toLowerCase())

);

}

rechercherParGenre(genre: string): Livre[] {

return this.livres.filter(livre =>

livre.genres.includes(genre)

);

}

obtenirStats(): { total: number; auteurs: Set<string> } {

return {

total: this.livres.length,

auteurs: new Set(this.livres.map(l => l.auteur))

};

}

}

*// Utilisation*

const biblio = new MaBibliotheque("Bibliothèque Centrale", "123 Rue des Livres");

biblio.ajouterLivre({

titre: "1984",

auteur: "George Orwell",

annee: 1949,

genres: ["Dystopie", "Fiction"]

});

biblio.ajouterLivre({

titre: "Le Petit Prince",

auteur: "Antoine de Saint-Exupéry",

annee: 1943,

genres: ["Conte", "Philosophie"]

});

### TD2.2 : Fonctions et Generics

typescript

*// Exercice 1 : Fonctions utilitaires*

function filtrer<T>(

tableau: T[],

predicat: (element: T) => boolean

): T[] {

return tableau.filter(predicat);

}

function mapper<T, U>(

tableau: T[],

transformation: (element: T) => U

): U[] {

return tableau.map(transformation);

}

function reduire<T, U>(

tableau: T[],

accumulateur: (acc: U, element: T) => U,

valeurInitiale: U

): U {

return tableau.reduce(accumulateur, valeurInitiale);

}

*// Exercice 2 : Classe générique pour un cache*

class Cache<K, V> {

private storage = new Map<K, V>();

private maxSize: number;

constructor(maxSize: number = 100) {

this.maxSize = maxSize;

}

set(key: K, value: V): void {

if (this.storage.size >= this.maxSize) {

const firstKey = this.storage.keys().next().value;

this.storage.delete(firstKey);

}

this.storage.set(key, value);

}

get(key: K): V | undefined {

return this.storage.get(key);

}

has(key: K): boolean {

return this.storage.has(key);

}

delete(key: K): boolean {

return this.storage.delete(key);

}

clear(): void {

this.storage.clear();

}

size(): number {

return this.storage.size;

}

}

*// Utilisation*

const userCache = new Cache<number, Utilisateur>(50);

userCache.set(1, {

id: 1,

nom: "Dupont",

email: "dupont@email.com",

dateCreation: new Date()

});

### TD2.3 : Système de gestion d'étudiants

typescript

*// Types et interfaces*

type Niveau = "L1" | "L2" | "L3" | "M1" | "M2";

interface Note {

matiere: string;

note: number;

coefficient: number;

}

interface Etudiant {

id: number;

nom: string;

prenom: string;

niveau: Niveau;

notes: Note[];

}

*// Classe de gestion*

class GestionEtudiants {

private etudiants: Map<number, Etudiant> = new Map();

private prochainId: number = 1;

ajouterEtudiant(

nom: string,

prenom: string,

niveau: Niveau

): Etudiant {

const etudiant: Etudiant = {

id: this.prochainId++,

nom,

prenom,

niveau,

notes: []

};

this.etudiants.set(etudiant.id, etudiant);

return etudiant;

}

ajouterNote(

etudiantId: number,

matiere: string,

note: number,

coefficient: number = 1

): void {

const etudiant = this.etudiants.get(etudiantId);

if (!etudiant) {

throw new Error("Étudiant introuvable");

}

if (note < 0 || note > 20) {

throw new Error("La note doit être entre 0 et 20");

}

etudiant.notes.push({ matiere, note, coefficient });

}

calculerMoyenne(etudiantId: number): number {

const etudiant = this.etudiants.get(etudiantId);

if (!etudiant || etudiant.notes.length === 0) {

return 0;

}

const sommeNotes = etudiant.notes.reduce(

(acc, n) => acc + (n.note \* n.coefficient),

0

);

const sommeCoefficients = etudiant.notes.reduce(

(acc, n) => acc + n.coefficient,

0

);

return sommeNotes / sommeCoefficients;

}

obtenirClassement(): Array<{ etudiant: Etudiant; moyenne: number }> {

const resultats = Array.from(this.etudiants.values()).map(etudiant => ({

etudiant,

moyenne: this.calculerMoyenne(etudiant.id)

}));

return resultats.sort((a, b) => b.moyenne - a.moyenne);

}

rechercherParNiveau(niveau: Niveau): Etudiant[] {

return Array.from(this.etudiants.values())

.filter(e => e.niveau === niveau);

}

exporterJSON(): string {

return JSON.stringify(

Array.from(this.etudiants.values()),

null,

2

);

}

}

## 🚀 TP 2 : CONVERSION JS EN TS + MINI-PROJET

(voir TP2 HTML)

typescript

const sortedTasks = taskManager.sortTasks(

taskManager.getAllTasks(),

'priority',

'desc'

);

sortedTasks.forEach(task => {

console.log(`${task.title} - ${TaskUtils.getPriorityLabel(task.priority)}`);

});

*// Exporter les données*

console.log("\n=== Export JSON ===");

const jsonExport = taskManager.exportToJSON();

console.log(jsonExport);

*// ========================================*

*// TESTS UNITAIRES (Optionnel mais recommandé)*

*// ========================================*

function runTests(): void {

console.log("\n=== TESTS ===\n");

*// Test 1: Création de tâche*

try {

const tm = new TaskManager();

const task = tm.createTask("Test", "Description test", PriorityLevel.LOW);

console.assert(task.id === 1, "❌ Test création échoué");

console.log("✅ Test création de tâche réussi");

} catch (error) {

console.log("❌ Test création échoué:", error);

}

*// Test 2: Titre vide*

try {

const tm = new TaskManager();

tm.createTask("", "Description");

console.log("❌ Test titre vide échoué (devrait lever une erreur)");

} catch (error) {

console.log("✅ Test titre vide réussi (erreur attendue)");

}

*// Test 3: Changement de statut*

try {

const tm = new TaskManager();

const task = tm.createTask("Test", "Description");

tm.changeStatus(task.id, TaskStatus.DONE);

const updatedTask = tm.getTask(task.id);

console.assert(

updatedTask?.status === TaskStatus.DONE,

"❌ Test changement statut échoué"

);

console.assert(

updatedTask?.completedAt !== undefined,

"❌ Test date complétion échoué"

);

console.log("✅ Test changement de statut réussi");

} catch (error) {

console.log("❌ Test changement statut échoué:", error);

}

*// Test 4: Filtrage*

try {

const tm = new TaskManager();

tm.createTask("Task 1", "Desc 1", PriorityLevel.HIGH);

tm.createTask("Task 2", "Desc 2", PriorityLevel.LOW);

tm.createTask("Task 3", "Desc 3", PriorityLevel.HIGH);

const highPriorityTasks = tm.filterTasks({ priority: PriorityLevel.HIGH });

console.assert(

highPriorityTasks.length === 2,

"❌ Test filtrage échoué"

);

console.log("✅ Test filtrage réussi");

} catch (error) {

console.log("❌ Test filtrage échoué:", error);

}

*// Test 5: Suppression*

try {

const tm = new TaskManager();

const task = tm.createTask("Test", "Description");

const deleted = tm.deleteTask(task.id);

const found = tm.getTask(task.id);

console.assert(deleted === true, "❌ Test suppression échoué");

console.assert(found === undefined, "❌ Test suppression échoué");

console.log("✅ Test suppression réussi");

} catch (error) {

console.log("❌ Test suppression échoué:", error);

}

console.log("\n=== FIN DES TESTS ===\n");

}

*// Exécuter les tests*

runTests();

export {

TaskManager,

TaskUtils,

PriorityLevel,

TaskStatus,

Task,

TaskFilter,

TaskStats

};

### Partie 2 : Interface HTML avec TypeScript

(Voir .ts)