**Classes in TypeScript**

We know that TypeScript is an object-oriented programming language. The class keyword is used to declare a class in TypeScript. An instance of the class can be created using the new keyword.

Let's take a look at a simple class example:

class Greeting {

//fields

name: string;

//constructor

constructor(name: string) {

this.greeting = name;

}

//methods

greet() : string {

return "Hello, " + this.name;

}

}

//creating an object

let greeter = new Greeting("James");

greeter.greet(); //returns "Hello, James"

**Inheritance**

In TypeScript, we can implement an inheritance hierarchy using the extends keyword, similar to other object-oriented programming languages.

class Animal {

move(distanceInMeters: number = 0) {

console.log(`Animal moved ${distanceInMeters}m.`);

}

}

class Dog extends Animal {

bark() {

console.log('Woof! Woof!');

}

}

const dog = new Dog();

dog.bark(); // returns 'Woof! Woof!'

dog.move(10); // returns 'Animal moved 10m.'

**Access Modifiers**

* **public** - In TypeScript, each member is public by default. We can still mark a member public explicitly.
* **private** - We can mark a member as private that cannot be visible outside of the class.
* **protected** - If a member marked as protected, then the member can be accessed only by its containing class and deriving classes.

**Readonly modifier**

We can make properties accessible but immutable by using the readonly keyword. Readonly properties must be initialized at their declaration or in the constructor.

class Employee {

readonly name: string;

readonly dept\_id: number = 123;

constructor (name: string) {

this.name = name;

}

}

let e1 = new Employee("Grace");

e1.dept\_id = "543"; // error! dept\_id is readonly.

**Modules**

In TypeScript, the code we write is globally scoped by default. TypeScript provides modules and namespaces to restrict scopes and also to organize and maintain a large codebase. All variables, classes, and functions declared in a module are not accessible outside the module. A module is created using the export keyword and used in another module using the import keyword.

To export a class, function or variable, add the export keyword at the begining.

// 'module.ts' file

export function sayHello(){

console.log("hello");

}

export class Employee{

empCode: number;

empName: string;

constructor(name: string, code: number) {

this.empName = name;

this.empCode = code;

}

displayEmployee(){

console.log(`Employee Code: $ {this.empCode} , Employee Name: ${this.empName} `);

}

}

export const maxLength : number = 1200;

The syntax for importing a module: import { export name } from "file path without extension";. For example , import { Employee } from "./module";

//'main.ts' file

//importing the Entire Module into a Variable

import \* as Emp from "./module";

console.log(Emp.maxLength); // returns '1200'

Emp.sayHello(); //returns 'hello'

let empObj = new Emp.Employee("Gavin" , 2);

empObj.displayEmployee(); // returns 'Employee Code: 2 , Employee Name: Gavin'

**Accessors and Mutators**

TypeScript supports getter and setter methods to access and set class members. The getter and setter methods are created using the get and set keywords.

class MyClass {

private \_width: number;

private \_height:number;

get area() {

return this.\_width \* this.\_height;

}

set width(newWidth : number){

console.log("setting width for square...");

this.\_width = newWidth;

}

set height(newHeight : number){

console.log("setting height for square...");

this.\_height = newHeight;

}

}

let obj = new MyClass();

obj.width = 10;

obj.height = 5;

console.log("area: " + obj.area);

//output will be:

//setting width for square...

//setting height for square...

//area: 50

**References**

* [TypeScript Docs - Classes](https://www.typescriptlang.org/docs/handbook/classes.html)