Typescript

problems with javascript language

- being dynamic, often leads to more TypeError
- complete OO programming concepts missing like Interfaces, custom types

Typed JavaScript at Any Scale.

TypeScript extends JavaScript by adding types to the language. TypeScript speeds up your development experience by catching errors and providing fixes before you even run your code.

https://www.typescriptlang.org/

Handbook - The TypeScript Handbook

Over 20 years after its introduction to the programming community, JavaScript is now one of the most widespread cross-platform languages ever created. Starting as a small scripting language for adding trivial interactivity to webpages, JavaScript has grown to be a language of choice for both frontend and backend applications of every size.

https://www.typescriptlang.org/docs/handbook/intro.html

Type script Interfaces

```
// brand
var MRFWheel = /** @class */ (function () {
    function MRFWheel() {
        console.log("MRFWheel instance created..");
    }
    MRFWheel.prototype.rotate = function () {
        console.log("MRF-wheel rotating..");
    };
    return MRFWheel;
}());
// brand
var CEATWheel = /** @class */ (function () {
    function CEATWheel() {
        console.log("CEATWheel instance created..");
    }
    CEATWheel.prototype.rotate = function () {
        console.log("CEATWheel rotating..");
    };
    return CEATWheel;
}());
/*

    design & performance issues
    ---------------------------
    -> dependent & dependency are tightly-coupled
       => can't extend module with new features easily
    -> on every move, we creating & destroying new wheel instance
```

```
       ==> resource use high i,e slow
    -> unit testing not possible
       ==> dev/bug fix slow


       reasons for above these issues ?

       ==> dependent creating it's own dependency in it's home ( class | method )


       solution :

       ==> never create dependency in dependent's home, inject from outside  ( Dependency inversion principle ) Inversion of co


       ------------------------------------------------------------------------

       OO principles a.k.a SOLID principles


        S: Single-responsibility principle
        O: Open for entesions-closed for modification principle
        L: Liskov substitution principle
        I: Interface segregation principle
        D: Dependency inversion principle

-------------------------------------------------------------------------

*/
var Car = /** @class */ (function () {
    function Car(wheel) {
        this.wheel = wheel;
    }
    Car.prototype.setWheel = function (wheel) {
        this.wheel = wheel;
    };
    Car.prototype.move = function () {
        this.wheel.rotate();
        console.log("Car moving..");
    };
    return Car;
}());
var mrfWheel = new MRFWheel();
var ceatWheel = new CEATWheel();
var car = new Car(mrfWheel);
car.move();
car.move();
```