Subject :

Year. 2022 Month. 12 Date. 15

Typescript : The Complete Developer's Guide

01 getting started with typescript

01 how to get help

How to get help, twitter: @ste-grider

02 typescript overview

what is typescript? typescript = js + type system, still : writing JS, adding: additional syntax, type system: 1. help: catch errs -> development, ~~->~~ usual JS: execute, 2. type annotation -> ts compiler, type annot: like comments, js: only: active: ~~during~~ during dev -> browser: no idea, at last: get a js file, 1. performance optimization: X,

TS code: JS with annotation -> ts compiler: cmd tool -> plain JS code -> browser,

me: execute TS X, execute JS √, typescriptlang.org/play => show ts, JS: side by side,

Summary: ts: same: js + extra doc, ts: no effect: code: get executed, ts: friend: sitting behind you: while coding

Subject :

Year. Month. Date.

03 environment setup

set our local machine ! work with ! ts, npm install -g typescript ts-node, one time setup, we assume ! you have ! node & npm installed, note ! if err : run with sudo, now ! tsc : typescript compiler, tsc ! compile → plain js, installing vscode instructions,

04 a first app

ready ! working on our project, our project : network req → fetch some data → print,

steps ! take look ! api ! used for fetching data, create new proj dir, cr package.json, install axios ! make req, write code, take look ! api ! jsonplaceholder.typicode.com ! Fake JSON api, 100 todos ! no ! need ! all items ! just first ! jsonplaceholder.typicode.com/todos/1, mkdir fetch-json, cr package.json ! manage deps, npm init -y, npm install axios ! smell ! run rather quickly

05 executing typescript code

ALVAND create file ! index.ts, ts ! short for ! typescript, goal ! fetch data → print

Subject :

Year. Month. Date.

import axios from 'axios'; make : get req => axios.get(url) ; async => return promise,

response => log(response.data), how to run? browser directly X, using node X,

compile -> plain JS, tsc index.ts => index.js : created, node index.js, compile & run:

gets boring : module : ts-node : compile and execute, ts-node index.ts,

06 one quick change

console.log(response.data) => ugly, extract : id, title, completed, print nicer, part of

const finished = todo.finished

data prop, assign to another temp var, const ID = todo.ID, const title = data.Title,

when logging them : all : undefined => what's wrong?

07 catching errors with typescript

ID, Title, finished => X actual name : id, title, completed, we don't know this bug

until executed, typescript : helps us, catch these errors, if we have : comment : data

property : id, title, completed => using typescript : interface, interfaces : define

ALVAND structure obj, ignoring : userId => we can do this, response.data as Todo

in

Subject :
Year. Month. Date.

now all: misspelled: compile err, good of ts: help us catch errors, during development,

08 Catching more errors

function logTodo(id, title, completed), logTodo(id, completed, title); mispelling:

find error: until run code, function todoLog(id:number, title:string, completed:boolean){ ... }

now, boolean → string: compile error, we saw: two examples: of: most common errors,

both case: run: find error; typescript: complite time

2022/12/15

02 What is a Type System

01 do not skip course overview

course structure, do not skip: how you: approach & learn: typescript, Syntax +

Feetures: what is: interface?, what: syntax: define, interface?, Vs: Design Patterns:

how to write: reuseble code, our mein focus, first: focus on syntax & features → projects:

ALVAND focusing on: Design Pattern,

Subject :

Year. Month. Date.

02 types

first topic: basic types in ts, for each topic: plan def + overview -> why do we care? -> examples

-> when to use, type: easy way, value: properties & functions has, array: type, object: type, "red": a string, a value: all properties & methods: string has, string: charAt, concat(), ..., shortcut: properties & functions: here: is a string, every value: in ts: has a type,

03 more on types

types: string, number, boolean, ~~date~~, Date, Todo, not restricted: this types: every value: has: type, Types: primitive, object types, primitive: number, boolean, void, undefined, string, symbol, null, object types: any ~~type~~ types: we create, some for functions, arrays, classes, objects, why divide? trick ts compiler: with object types, think: one type: is other, why we care? we declared Todo: id, title, completed, => only these properties, ts compiler -> only for our code, other engineers: labeling types => understand our code

ALVAND

Subject :

Year. Month. Date.

04 exemples of types

entire idea : behind : type system : easy way : refer : properties & functions, today = new Date(),

today. => auto popup : tell methods : based on Date, const person = { age : 23 } =>

hover : person : show all info, person. => shows age, class Color {}, const red = new

Color(), => hover red => type Color,

05 where do we use types

we got : def, why care, exemples, => where we use ? : every where,

2022/12/15

03 Type Annotations in Action

01 type annotations and interfaces

type annotations & type inference, veriables → functions → objects, Type Annotation :

we write, what type : is veriable, Type Inference : typescript : tryies : figure out :

ALVAND type, Type Annot ⟷ Type Inference

↳ we tell typescript ↳ Typescript guesses

Subject :

Year. Month. Date.

02 annotations with variables

snippet: figure out: type annotation: variables, create: variables.ts, const apples:

number = 5 => we only assign: type number → apples, apples = true: error, with

every type: primitive, object: we can use types, let speed: string = 'fast'; let nothing-

much: null = null, let nothing: undefined = undefined, or with Date ...

03 object literal annotations

we can use for: let colors: string[]; => can contain: nothing but strings, this: strings

: not going: create array, just says: type, one complicated: figure out: type annotation

and initialization, also with classes, class Car {}. let car: Car = new Car(), object

literals: let point: {x: number; y: number} = {x: 10, y: 20}, point: only with number

properties, x, y,

04 annotations around functions

ALVAND for functions: somehow nesty, what different: arguments, values: return

Subject :

Year. Month. Date.

const logNumber: (i: number) => void = (i: number) => { console.log(i) },

annotation function

let apples: number = 5 <=> let apples = 5 : both; when hover: apples: number,

05 understanding inference

const color = 'red'; if decl & init : same line -> typescript ; figure out type,

var Decl var Initialization

if: let apples; apples = 5 => typescript : can not figure out, apples: any, when : Type

Inference : always! => 80% of code: let apples = 5; rarely : let apples: number = 5;

Three scenarios : use explicit type

06 the any type

three scenarios: first: function: returns: 'any' type, clarify value, let coordinates =

JSON.parse(json), coordinate: any, JSON.parse: input: 'false' -> boolean, '4' -> number

'{"value": 5}' -> {value: number}, input: variety of types, output: same as input,

JSON.parse: can not predict, JSON.parse: I can not guess output type: too complicated,

ALVAND I will use: any, any; I have no idea, what is real type, avoid vars: any type

Subject :

Year. Month. Date.

if typescript knows type: acts to suggest functions, does not know: no error checking,

07 fixing the any type

how to fix? add type annot: coordinates: { x: number; y: number; },

08 delayed initialization

second case: dec one line, init: other line, let words = ['green', 'blue', 'yellow'], let foundWord; => if word == green: foundword = true, foundword: implicitly gets: any =>

let foundWord: boolean;

09 when inference does not work

third case: we want a veriable: type: can not: inferred, scenario: let numbers = [-1,0,3]

let numberAboveZero; => if found, assign it, else: assign false, let numberAboveZero = false

if (num[i] > 0) : numAboveZero = num[i] -> error, numberAboveZero: boolean | number,

we looked: three cases

ALVAND

Subject :

Year. 2022 Month. 12 Date. 15

04 Annotations with Functions & Objects

01 more on annotations around functions

Type Annotations & Type Inference: variables, function, objects => now applied: functions,

you may thought: we talked: functions: before => const logNumber: (i: number) => void =

(i: number) => { log(a) }, Type Inference: for functions → typescript: tries: figure out:

type: function will return, cons add = (a,b) => {} : warning! type: a: any, => err:

const add = (a: number, b: number) => {}, (a:number, b:number): number {} => non void, should

return a number,

02 inference around functions

we are going: annotate: every: argument, and return annotation => immediately!

analyze: body function: see: if we returned, typescript: just check return type, not logic

of our: function, add: a-b: no err, no type inference: for arguments: always

add annot, hover to add: see annotation, type inference: only about: return value: function,

Subject :

Year. Month. Date.

const subtract = (a:number, b:number) => { a-b; }: no return statement, return value : becomes: void, so : always : add annotation,

03 annotation for anonymous functions

Look around : some alternative syntax, function divide(a:number, b:number): number,

same for anonymous function : const multiply = function(a: number, b: number) : number,

arrow function, named function, anonymous function, same,

04 void and never

one last thing to mention, special types, const logger = (msg : string): void => log,

void: not return anything, can return null or undefined or : return, const throwError

= (msg:string) : never => { throw new Error(msg) }, never: we will never reach end of

this function, if you want to : return anything : don't put never: const throwError

= (msg: string): string => { if (!msg) throw new Error(msg) return msg },

Subject :

Year. Month. Date.

05 destructing with annotations

suppose we have: const forecast = { date: new Date(), weather: 'sunny'}, const logWeather = (forecast : {date: Date; weather: string}) : void => {log(...)}

ES 2015: {date, weather}, destructing: replace variable name: fields:

forecast → {date, weather}

06 annotations around objects

until now, variables & functions, let we went to talking about: object destructring,

const profile = { name: 'alex', age: 20, coords: { lat: 1, lng: 2}, setAge(age: number) { this.age = age}}, const {age} = profile or: const {age}: {age: number} = profile,

const {coords: {lat, lng}} → look out at coords property, try to pull out lat and lng,

const {coords: {lat, lng}} = profile, for annotation we have: const {coords: {lat, lng}}: {coords: {lat: number; lng: number}} = profile, double up annots when destructuring, functions: you rely on annotation, variable: as much as: Ts: inference

Subject: Typescript: The Complete Developers Guide

Year. 2022 Month. 12 Date. 15

01 Getting Started with Typescript

01

This section talks about how to get help, there are several ways to access to stephan grider, udemy website is one way, also stephan twitter address is: @ ste_grider

02

we want to have a quick overview of what is typescript, typescript is: JS + Type System, note that you still write JS code in TS, but there is additional syntax. Type system has some advantegous: 1. helps you catch errors during development, but in usual JS, you have to execute the program to find out the errors. 2. we have type annotation, that ts-compiler uses, type annotations are like little comments, that compiler checks them to be correct. 3. typescript is only active during development, and browser has no idea of TS, when you compile TS file, at last you get a js file, that use with browser. note that there is no performance optimization with TS, we can say: TS is a JS file with annotation => TS-Compiler: is a Command line

Subject :
Year. Month. Date.

tool, that gives us plain JS from TS code. => at last we give plain JS file to the browser. note that we do not execute TS file, we execute JS file.

There is a site: typescriptlang.org/play that shows equivelent of ts code side by side by its JS. At last, and as summary we can say that ts is just entire documentation, and ts has no effect on code; its just like a friend sitting behind you, while your code.

23,04

We went to set our local machine, so we can use ts, we assume you already have node and npm installed on your machine, you should enter: npm install -g typescript ts-node, this is a one time instalation. If you got permission denied error, you can should use that command with sudo. That commands will install typescript and give you access to tsc: typescript compiler and also installs ts-node, which compiles typescript code to JS and then executes it with node.

ALVAND Now we have everything setup, we can work on our first project.

Subject :

Year. Month. Date.

Our project is as follows: we make a network request, then fetch some data, and print it. Our steps are as follows: first we take a look at api used for fetching data, then create a new project directory, in it, we create package.json with command npm init -y, package.json is responsible for maneging our dependecies. Now we install axios with : npm i axios, we make get req to jsonplaceholder.typicoder.com/todos/1, this site contains JSON that you can get. /todos contains 200 todo item, we dont need all of them, just first one: /todos/1.

05

we create the file index.ts, ts is short for type script, note our goal is to fetch some data and print it. we import axios by : import axios from 'axios', then we make a get request by : axios.get(url), this is a async command and returns a promise, then : (response) => log(response.data), so how to run this file? we can not use browser or use nod directly, we first compile it by :

ALVAND tsc index.ts, then it gives: index.js, then we run it with node.

Subject :

Year. Month. Date.

because this work is boring, there is a utility called ts-node that does both works.

06

If we ~~print~~ print response.data by : console.log(respanse.data), it is ugly, so we went to extract id, title and completed and print them prettier. So we pull data prop and assign them to another temp variable. (todo) => const ID = todo.ID, const title = data.Title, const finished = todo.finished, we get all of them as undefined, so what's wrong?

07

The problem is: we have none of these fields: ID, Title, finished, their actual names are: id, tittle, completed, note that we can not get this bug until we execute our code. We use TS to help us catch these errors at compile time. If we had a comment that tells us how proper fields look like, we would not make that problem again. we use typescript interface, interfaces define structure of objects.

ALVAND with ignoring userId, we can do this: const todo = respons.data as Todo

Subject :

Year. Month. Date.

now all misspelled fields will produce compile time error.

08

we can create a function: function logTodo(id, completed, title) and invoke it by:

logTodo(id, title, completed); although we misplaced title and completed, we can not figure out until we run the code, instead we can use TS to at least check type of arguments: function logTodo(id: number, title: string, completed: boolean) {...}

Now if we put boolean → string, we get compile time error. we saw two examples of most common erross, both cases we get at run time, but now we get at compile time.

2022/12/18

02 What Is a Type System?

01

This section is about course structure, please Do Not Skip It! This tells you how ALVAND to approach and learn typescript. There is Syntax + Features, for example

Subject :
Year. Month. Date.

we say, what is an interface? and what syntax to define it? vs the Design patterns, they say: how to write reusable code, this is our main focus, first, we focus on syntax & features and then in projects, we focus on Design Patterns.

02

Our first topic is: Types in TS, for each topic, we have plenty of definitions + overview and then why do we care section, this contains examples that show us when to use this feature. Type is an easy way to show what properties and functions a value has. for example we have array type, object type, string type: "red", a string value, has all properties and methods that string has, for example: charAt(), concat(), ... so type is a shortcut to say what properties & functions a value has. Also note: Every value in TS has a type.

03

Some types are: string, number, boolean, Date, Todo and we are not restricted to these types, every value has a type, Types are: primitive & object types,

ALVAND

Subject :

Year. Month. Date.

primitives are: number, boolean, void, undefined, string, symbol and null. Object types are any types we create, some for functions, arrays, classes and objects.

So why we divide them? so we can trick the ts compiler with object types, think one type is other, why we care? we declared Toda with: id, title, completed; so it has only these properties, ts compiler analyses our code, and other engineers can understand our code, with labeling types.

04, 05

Entire idea behind type system is an easy way to refer to properties and functions a value has. if you type: today = new Date(); and then type today. => something will popup and ten methods based on Date. const person = {age: 20} => if you hover on person, it shows all info about person. class Color {}, const red = new Color(); => if you hover on red, it shows it type is Color.

So you may ask: where we use types? answer is everywhere.

Subject :

Year. 2022 Month. 12 Date. 18

03 Type Annotations in Action

01

we want to talk about type annotations & type inferences ; we talk about them in this order: variables, functions, objects . Type Annotation is : we write what type, a variable is . Type Inference is when typescript tries to figure out what is the type of this variable. So : Type Annotation : we tell ts <=> Type Inference: ts tells us.

02

we want to see annotation, with variables, we create variables. ts file, then : const apples : number = 5 ; we can only assign type number to apples, so if we say : apples = true ;→ that's compile time error. with every types : primitives & objects we can use types. let speed : string = 'fast' ; let nothingMuch : null = null ; let nothing : undefined = undefined .

ALVAND

Subject :

Year. Month. Date.

03

We want to see object literal annotations; we can use it for: let emrs: string[];

this can only contain strings; note that this syntax is not going to make

a string array; it specifies only the type. One complicated thing is to see where

is type annotation, and where is initialization; also with classes we have:

class Car{}; let car: Car = new Car(); for object literals we have: let point = {x: numb

; y: number} = {x: 10, y: 20}; this creates point only with number properties

x & y.

04

we want to see annotations around functions; for functions it is somehow nesty,

what different arguments, and what value returns, const logNumber: (i: number)

annot

=> void, = (i: number) => {console.log(i)},

function init

05

const color = 'red' => if declaration and initialization are on the same line;

(

ALVAND var Dec var initialization

Subject :

Year. Month. Date.

typescript can figure out type ; but if : let apples; apples = 5 => typescript connot figure out the type, and apples gets any type. So when we have type inference with veriables?? always!! => you will see a lot of code: let apples = 5; but rarely see ; let apples: number = 5; But there is three scenerios that you use implicit ty.

06

we want to consider that three scenarios: 1st: function returns 'any' type and we need to clerify the returned value : let coordinates = JSON.parse(json) ; now coordinates gets 'any' type. we expect from JSON.pevse that for these inputs, return that outputs : 'false' → boolean, '4' → number, '{"value":5}' → {value: number}

But json.parse can not determine the output ; its too complicated for it, so it will use 'any' type. any means ; I have no idea what is real type. we should avoid variable with 'any' type. If typescript knows type ; it auto suggests ALVAND functions but when it doesnot know, no error checking is performed.

Subject :

Year. Month. Date.

07,08

So how can we fix the any type? we can fix it by adding type annotations;

for coordinates we have: {x: number; y: number}. Our 2'nd type is declaring

on one line and initializing on other line. For example we have an array: let

words = ['green', 'blu', 'yellow']; we have a variable foundWord, we want to if

find 'green' in words: foundWord = true; but if we say let foundWord; it implicitly

gets any type. So we should say: let foundWord: boolean;

09

Now we consider our 3'rd case, we want a variable type that can not be inferred,

For example supose this scenario: let numbers = [-10, -1, 12] let numberAboveZero:

false

we want to if found a number above zero, assign it to this var, and if we

didn't; assign false to it. So: if (num[i] > 0) numAboveZero = num[i] => gives Err,

because it thinks numberAboveZero is boolean, So we should say: numberAboveZero:

boolean | number. So we looked at our three cases.

ALVAND

Subject :

Year. 2022 Month. 12 Date. 13

04 Annotations with Functions & Objects

01

we said we will consider Type Annotations & Type Inference for; variable, functions and objects. we talked about variables, you may think we talked about functions also when we said; const logNumber : (i:number) => void = function (i:number) => {log(...)}. for type inference for functions, typescript tries to figure out the type function will return, const add = (a,b) => {...} ; if we code this, we get a warning ; type a is 'any' so we should write; (a:number, b:number) :number => although this gets error if we do not return anything in body ; non void should should return a number.

02

we went to see how inference works around functions more concisely. we are going to annotate every argument and return value. so ts compiler analyzes our body of function to see if we returned the correct type. note that ts only checks

Subject :

Year. Month. Date.

return type not logic of our function. So if in odd function we return a+b that is ok. There is no type inference for arguments, so we have to always add annotation. types inference is only about return value in functions; but we should be careful for example: const subtract = (a:number, b:number) => {a-b;} => there is no return statement so it inferences void; we conclude that we should always add return type annotation.

03

Annotation for anonymous functions is same: function divide(a:number, b:number): number => const multiply = function(a:number, b:number): number. This is one of these: arrow function, anonymous function.

04

One last thing to mention is special types, const logger = (msg: string): void

void means it is not returning anything. we have also null or undefined:

ALVAND const throwError = (msg: string): never => {throw new Error(msg)}

Subject :
Year. Month. Date.

never says we will never reach end of this function; if we went to return anything, don't put never: const throwError = (msg: string): string => { if (!msg) throw new Error(msg) return msg }

05 }

Suppose we have: const forecast = { date: new Date(), weather: 'sunny' }

const logweather = (forecast: { date: Date; weather: string }): void => { log(...) }

as in ES 2015; we can destruct and replace variable name in fields: forecast → { date, weather }

06

We went to consider annotations around ~~functions~~ objects, until now we checked variables & functions. Let we went to talk about is object destructing,

Suppose we have: const profile = { name: 'alen', age: 20, coords: { lat: 10, lng: 20 }, setAge(age: number) { this.age = age } } Pause Lock of paper

Subject

Year: 2022 Month: 12 Date: 18

05 Mastring Typed Arrays

01 arrays in typescript

Arrays in ts, quick Def: plain JS Array ; push, pop, map, foreach, one difference : stick one type, only strings, one specific type: in : array , create : arrays.ts ,

we put type annotation : when : empty array : const carMakers : string[] = [] ; if : ~~const carMakers = [] ;~~ const carMakers = [] : 'any' , for 2d : string[][] ,

02 why typed arrays

why care? why seperate topic ?, some advantageous, some disadvantageous ,

ts : knows : carMakers : array of string , const car = carMakers[0] => hover : car : string ,

also works : carMakers.pop() , also : carMakers.push(100) => Err : incompatible types

when declare : array : get help : 'map', 'foreach', 'reduce' , carMakers.map((car:string) :

string => { return car;} or : return car.toUpperCase => gives auto complete ,

03 multiple types in arrays

how : use a arrays => multiple types : in them, const importantDates = [new Date, '2039-10-12']

=> type: (string|Date)[]

04 when to use typed arrays

any time : collection of records, with similar types, tuples: very similar: arrays,

see : what differences : tuples & arrays

2022/12/18

06 tuples in typescript

01 tuples in typescript

Tuple : Array-like structure, each element: represent: some: property record, inside

tuple : min a lot of types, { color: 'brown', carbonated: true, sugar: 40 }, one

object : one drink, represent: this object, array : ['brown, true, 40] => lost

Subject Year: Month: Date:

in array: we should: memorize, color, carbonated, sugar, difference: loss of info,

what is tuple: order matters, tuple: fixed order of elements,

02 tuples in action

create: tuples.ts, const pepsi = ['brown', true, 40] => (string|boolean|number)[]; we can swap elements, we want the order: break order → data model: breaks out,

add annotation: convert! array → tuple, pepsi: [string, boolean, number] = ['brown', true, 40]

rather than: repeat: [string, boolean, number] => type alias, type Drink = [string, boolean, number] => no array created, now: const pepsi: Drink = ..., re use,

we are not using tuples very often, tuples: not super useful,

03 why tuples

when to use? when working: csv, const carSpecs: [number, number] = [400, 3354]

looking this line: what are these numbers?, const carStats = { horsePower: 400, weight: 3354 }

Subject Year: Month: Date:

object : immediately : observable, type : much newer, what is going on?

2022.12.18

07 The all Important Interface

01 interfaces

Interfaces + Classes => strong code reuse, how : toget to : work together : micro

interface : create : new type, describe property names, value types => for object,

interface : new custom type,

02 long type annotations

create interfaces.ts, const oldCivic = { name : 'civic', year : 2000, broken : true },

printVehicle (vehicle : { name : string; year : number; broken : boolean })

long annotation

we are going to : duplicate this : for each fn,

03 fixing long annotations with interfaces

duplicate: long annotation, so: using interface, create interface → create new type: interface Vehicle {...} => V: uppercase, interface Vehicle { name: string, year: number, broken: boolean }, in order to call this: must have: name, year, broken }, ts: looped through: objects: ensured: have on that types,

04 syntax around interfaces

little note: about: interfaces, not limited: expressing: primitive values, any: type we want, also have function, interface Vehicle { name: string; date: Date; broken: boolean; summary(): string } => and objects: should have: all of this,

05 functions in interfaces

two more: pieces of syntax: do our vehicle: must have: all properties?

Subject Year: Month: Date:

if : interface vehicle { summary(): string}, const oldcivic = { name: 'civic',

year: new Date(), broken: true, summary(): string { return 'name'}}

=> oldcivic : satisfies : vehicle interface, Vehicle : only check : to see : summary,

any question : get asked : do you have summary?, additional properties: oldcivic:

does not matter, change Interface name: interface Reportable { summary(): string};

06 code reuse with interfaces

15

creating another object: const drink = { color: 'brown', carbonated: true, sugar: 40,

summary(): string { return 'my drink, has ${this.sugar} sugars'}, drink & oldcivic:

20

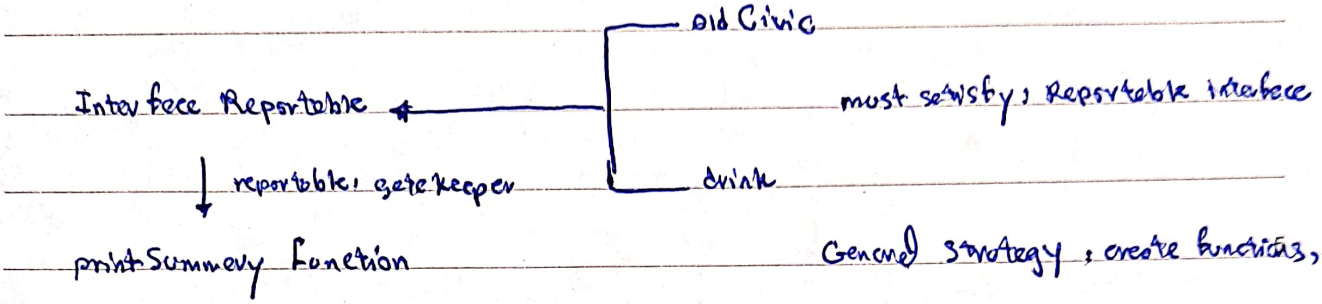
both: ~~same~~ summary() => both: type Reportable => use both with: prints-

ummary(reportable : Reportable), point: single interface, different object, encourage:

25

write generic: methods,

07 general plan with interfaces



accepts: args: type interface, objects/classes: decide: implement: given interface,

every application: we put together: using this, as long as: value: implement: interface:

can work with: function

15

20

25

Subject

Year: Month: Date:

5

10

15

20

25

Eiffel

Subject Year: 2022 Month: 12 Date: 20

05 Mastering Typed Arrays

01

we went to see typed arrays in TS. First we look at their quick definition, ts arrays are like plain JS arrays, they support: push, pop, map, foreach; there is only one difference: you should only stick on type in ts arrays. First we create arrays.ts. we should put type annotations when declaring empty array; for example we have: const carMakers: string[] = []; if we write const carMakers = []; Our array gets 'any' type. Also for 2d arrays we should write: string[][]

02

~~So~~ so why do we care about typed arrays, that we have created a separate section for them? they have some advantegous and some disadvantegous, ts knows that carMakers is an array of type string; So if we write: const car = carMakers[0] and hover on car, we see its type is string; it also works for: carMakers.pop()

and also writing carMakers.push(165) will give us: Err: incompatible values,

when declaring array we get help for its methods: map, forEach, reduce. we can

write: carMakers.map((car:string) => {return car}) or we can write: return car.-

toUpperCase => its gives us autocomplete on: toUpperCase.

10

03

we want to see how we can use arrays with multiple types in them. for example

we have: const importantDates = [new Date(), '2030-10-12'] ; it will give us 15

the type: (string | Date)[].

20

04

Any time we have a collection of records, with similar types, we should use typed

arrays. tuples are very similar to arrays, in the next chapter, we see their 25

differences.

Subject Year: 2022 Month: 12 Date: 20

06 Tuples in Typescript

07

Tuples are array like structure, each element represents some property record,

inside tuple, we can mix a lot of types. Suppose we have this object: { color: 'brown', carbonated: true, sugar: 40 }, this object represents one drink. We can represent this object as array: ['brown', true, 40] => by doing this we have lost info; in array we should memorize that the order is: color, carbonated, sugar, so we want to see what is tuple? in tuple, order matters, in tuple we have fixed order of elements.

02

So when to use tuples? they are not mostly used, but they are used when working with csv files, but suppose we have: const carSpecs: [number, number] = [400, 3354];

by looking at this line, we say: what are these numbers? but if we write:

const carStats = { horsePower: 400, weight: 3354 }; by looking at object, we can immediatly see what these fields are, but tuple is much harder to find out what is going on.

2022/12/20

07 The All Important Interface

07

In typescript, Interfaces + classes give us a strong code reuse, we want to get to how to work these two together nicely. interface creates new type, and describe property names, and value types for objects. As we said, interfaces are named custom types.

02

we create interfaces, its type, then: const oldCivic = { name: 'civic', year: 2000,

, broken: true}; then if we had a printVehicle method that takes old Civic es arg:

printVehicle(vehicle: {name: string; year: number; broken: boolean}) -> this is a

long annotation, so we are going to duplicate this for each car, and that is

a problem.

10

03

we want to fix long annotations with interfaces, we create an interface, so we create

a new type; interface Vehicle {...} so: interface Vehicle {name: string; years number;

broken: boolean}, not that first letter in interface is uppercase and the fields

can be divided by comma or semicolon, Now in order to call this, we must have 20

name, year and broken. when we pass an object as vehicle, ts loops through

objects and ensures we have that types. 25

04

There is a little note about interfaces, they are not limited to expressing primitive values, we can use them, with any type we want, they can also have functions, interface vehicle{ name: string; date: Date; broken: boolean; summary(): string} any object that wants to be vehicle, must have all these fields.

05

There is two more pieces of syntax, do our vehicle need all these properties? if we say: interface Vehicle { summary(): string} then: const oldCivic = { name: 'civic', year: new Date(), broken: true, summary() { return this.name}} , our oldCivic object satisfies the Vehicle interface; vehicle only checks to see summary function is implemented or not, the only question that gets asked is: do you have summary? additional properties in oldCivic does not matter, because vehicle now only checks for summary method, we chang its name to Reportable

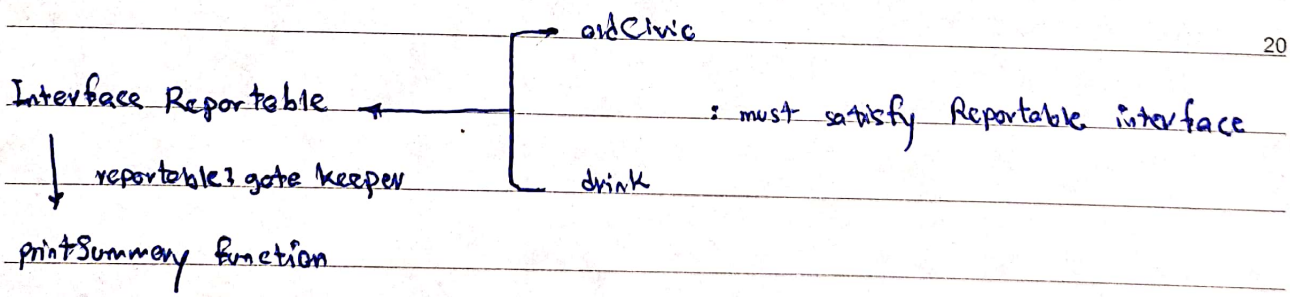
06

now we create another object, const drink = { color: 'brown', carbonated: true, sugar: 40, summery() { return 'My drink, has ${this.sugar} sugars' }}; now we have an interesting thing: drink & oldCivic both implement summery() method, so they are both of type: Reportable, we can use both with: printSummery(reportable: Reportable). So the point is with single interface, we can use different objects.

So this encourages us to ~~write~~ write generic methods that work with interfaces.

07

So our general plan with interfaces is:



General strategy: create functions that accept arguments as type interface; then objects/classes decide to implement that given interface. Every application we

put together, we use this. As long as a value implemented this interface, we can use it with that function.

Subject | Year: 2022 Month: 12 Date: 20

08 Building Functionality with Classes

01 classes

class: blueprint: create different objects: fields(values) & functions,

First: how: classes: work; functions, Second: add fields, create classes.ts,

class Vehicle { drive(): void { log(...) }} : blueprint, run this method: directly X

make instance :V, const vehicle = new Vehicle(); vehicle.drive(), this class:

also here: honk(): void { log(...) }, ts classes: little different: old JS classes

02 basic inheritance

looking: inheritance system, another class: Car, anything vehicle: do: Car

can do, copy peste on methods X, extends Vehicle: take on: different methods,

now: const car = new Car(); car.drive() or car.honk(), we can: optionally override:

parent methods -> inside Car: drive(): void { log('vroom') }, overriden,

Subject Year: Month: Date:

Vehicle : superclass or base class,

03 instance method modifiers

diff : btw : es2015, typescript, in : classes, modifiers : public, private, protected,

restrict : access, default : public, if you dont add : modifier : public, public!

called : anywhere, anytime, prive : other methods : exact same class, mak drive :

in class Car : private : const car = new Car() ; car.drive() => Err, make : startDri-

vingProcess() : void { this.drive() } in : Car class, now : car.startDrivingProcess(),

now : another : error => drive : Vehicle : public, drive ! Car : private => X, can not

change ! modifier : at child, private : restrict access : function, adding : private!

Case :

not adding : any application security layer, V have some method : don't want :

other developers : all, protected : just like : private, different : access method :

in : child classes, make, Vehicle → honk : protected => can only accessed :

Eiffel in Car

in class Car: super.drive => chugga chugga, this.driver => vroom ~~beep~~, But: super.honk => beep, this.honk => beep,

04 fields in classes

adding: two fields: to the: vehicle => these: are: actuall properties, name of property, then: annotation, color: string; => Err: not initialized, [mine: did not give err, until run], color: string = 'red', nicer: specify: color => when creating => const vehicle = new vehicle('orange') => we have to define constructor => constructor(color: string) { this.color = color? ; either: initialize: on same line, or inside constructor, Now: (inside vehicle): color: string; constructor(color: string) { this.color = color? shortcut: constructor(public color: string) => equivalent, we use: this shortcut, body constructor: even empty: we should put it, modifiers: effect: variables: same: functions,

Subject Year: Month: Date:

05 fields in typescript

we know: how fields: work in class, how about inheritance? Cv extends Vehicle

=> constructor => provide color string => error goes away, Cv: no constructor, constructor

in parent: called: automatically!, if: in Cv: constructor(public wheels: number){}

: error: must contain: super(), maybe we put: super('red'), we: dont want:

hardcode => constructor(public wheels: number, color: string): didn't put: public

before: color, => super(color), that explanations: was: real quick, because:

we will see: in the projects

20

06 where to use classes

classes: why care about? when to use? interfaces + classes: primary tool: in ts

25

different classes: work together: help of interfaces,

09 Design Patterns with Typescript

01 app overview

we said: we spend: lot of time: syntax: now finished, working: first application,

use ts feature: really reusable code, Randomly Generate: User : then: Company

show on map, every user/company: location, more familiar: classes, parcel-bundler

: ts executer, inside browser, sudo npm install -g parcel-bundler

02 bundling with parcel

mkdir maps, parcel-bundler → index.html → script of index.ts : parcel:

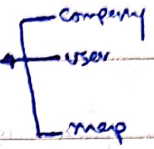
as soon as: sees: ts file: compile → JS, replace this script tag, maps >> src dir,

create index.ts: inside src, run the server by: parcel index.html

03 project structure

collection of different files, each file: class, class: on thing, Randomly generate User -> then on Company map

our classes: likely: user, company, map, src dir: index.ts



index.ts, User.ts: why uppercase: file: primary purpose: create & export a class, index.ts: not going: export any class, create class: user.ts: class User{}

04 generating Random data

user properties: user name, location: object: latitude, longtitude, we are going to: Randomly generate these props, initialization: inside constructor, faker: generate fake data, faker: take care of: all data: in our app, to access: module: we have just installed: import faker from 'faker': could not: find: declevations: module: faker,

05 type definition files

we installed faker module, see: error: could not find module, typescript code ↓ JS library

we can import code you and i wrote into typescript, but ts wants to know: ↓ JS library

types, all code we wrote in JS no type => solve: Type definition file, when installed:

amios: we didn't specify Type Definition files, we have to install type def. manually: if is not installed, once we find, we have to install type def file:

@types/libreryname? => @types/faker, Definitely Typed naming, these files:

entremely small, vast majority of time: you have to install type definition file,

06 using type definition files

if we hover on faker: press ctrl key: it becomes clickable link: index.d.ts

.d.ts: type definition file, inside type definition file: latitude & longitude: string.

in user class: doing some initialization, in user: lat&lng: object not initialized

automatically : log(location) : null or undefined, this.location.left = 72 => Err;

cannot : read : bt of undefined, to initialize leftString => initialize : location before,

parseFloat : take : string => return : number,

07 export statement in typescript

we : completed : our user, file : houses : single class, we are not going : actual code :

does anything with class, instead : export it, export class User {...}, index.ts:

import {User} from './user.ts', using : just export : when importing : use {},

safely import several, export default 'red' : user.ts, import red from './user',

or : import color from './user', in typescript : not usually use : default statements,

not use default exports : so always : put curly braces,

08 defining a company

repeat same process : create Company.ts, companyName : string, catchPhrase : string,

faker.address.latitude() : string → parseFloat(),

09 adding google map support

generate: new google dev project → enable support, google maps, → generate api key

→ add google maps, script: html file, got our api key: from: console, cloud.

google.com, add script: index.html, place the key=[], network → JS: js?key=

[] => status=200,

10 google maps integration

inside browser: console: google => { maps: {...}}, usually: install dependencies: using

npm, now: script -> directly: html file, global variable, without any import,

but if: type: google: Err: not found, does not understand: this global variable,

type definition: how third party (library), works, type def: not only: for npm modules, but also:

script tags: added in: directly: html file, @types/googlemaps, this going to tell ts: there is: global var: called google, and all properties & functions: it has, hover on google: namespace,

11 exploring type definition files

Map instance, new google.map.Map(): Err: wants two arguments: Element, options: opt? => optional, type: second arg: MapOptions, different options to customize map, ?: question mark: optional property, {zoom: 1}, also: center property: must satisfy: LatLngLiteral: ctrl+click on it: should have {lat: number; lng: number},

12 hiding functionality

focus on: index.ts, company: Reference companyName| catchPhrase | lat | lng
user : Reference name | lat | lng

googleMap: setZoom, setCenter, setStreetView, getHeading, ..., methods: we exposed in our index.ts, other engineer: on these: methods: break our app, ideal things, expose: map: addMarker, so, google map: and its dangerous functions: hidden, only things with custom map: create, add marker,

10

2022/12/21

13 why use private modifiers? here is why

to hide: create: custom map, capsulate google map, expose minimum amount of functionality,

create: CustomMap.ts, export custom map, hide: existance: google map, pretend: other engineers: can not see: inside this: file, googleMap: google.maps.Map; 20

default modifier: Map, marking: googleMap: private, now: create: user, access its properties, same for: Company, for CustomMap: use: addMarker, completly: eliminated:

need: global var: google.map => wrapped in CustomMap,

Subject Year: Month: Date:

14 adding markers

first: bad code: implementing : addMarker, Refactor: into a better approach,

addUserMarker, addCompanyMarker , import User, Company type, creating two 5

methods: to: addMarker: bad code, why? in just a little bit, for adding a Marker:

new google.maps.Marker ({ map: --, position: { lat: user.location.lat, ... ?? }, 10

15 duplicate code

15

completing: addCompanyMarker, addUserMarker & addCompanyMarker: look so similar,

what is bad here? two methods: ton of duplication, will be good: as long as

20

: you pass smt: has: lat & lng,

16 one possible solution 25

approach #1, addMarker (mappable: User() | Company), what does it do?

using or operator: ts: take a look: at two types; you can refrence: property if both user & Company: have that prop, result of or operator: limit properties, there is still: downside, what happen on future: if: CarLot -> display on map, or Park class -> for every new thing: a new import: user CarLot, Park, and: mappable: User | Company | CarLot | Park and.., a lot of adding, not great approach: not scalable, very thight coupling, btw our classes,

17 restricting access with interfaces

focus on: best solution, prev sol: custom map: depend upon: User, Company, CarLot, Park, rather than: custom Map: depending on classes: Hey User: you have to satisfy: Map requirements, invert dependecy, if you want to: have marker: to have location { let: number; lng: number} property, => make use of: interfaces, invert dependency: rather than: custom Map depending: other classes: user & Company:

interface Mappable ← User

↓ mappable gatekeeper, for addMarker ↑ must satisfy, Mappable, to work with;

addMarker method ↓ Company CustomMap

a value: can have multiple different type: User and implements Mappable 5

both,

10

18 implicit type checks

now: in customMap: no direct reference: to user or Company, now: customMap.

15

addMarker(user), customMap.addMarker(company), ts: no complains, behind

the scenes: inspecting: what type: user is, because user & company: have location

20

{ lat, lng } : x no Error, no need: extends (opposite of java), we never said: inside

user: should be: mappable, ts: did it: automatically,

25

19 showing popup window

click on marker: popup will come, make sure: hover marker: info showed,

Info window: is the popup, add event listener, adding popup: a bit complicated;

so: see the documentation, info window: wants a content: be displayed,

now: static text: 'Hi there!',

updating
20 ~~showing~~ interface definitions

customize: content: inside window, depend upon: passed user or company, in other

words: we want: mappable: responsible for: this content, adding new requirements;

for mappable, adding: markerContent(): string => to Mappable, implementing: in:

User & Company, descriptions: can be: html,

21 optional implements clauses

our app: feature complete, inside customMap: adding color property: to: Mappable

=> err: indents: in user & company, entire point of ts: help us find errors, root of

errors, indents not best place to see err msg, seeing errs in User & Company, inside customMap: export Reportable, then: in User: class User implements - Mappable, by adding implements, saying ts; helpus make sure: User implemented correctly, now: still: seeing err msg: inside indents also: in User, implements: not required: at all, reason: fail to: properly: implement: interface: get to true src of err,

15

22 app wrepup

reviews: important concepts: we learned, if: other engineer: open up: indents:

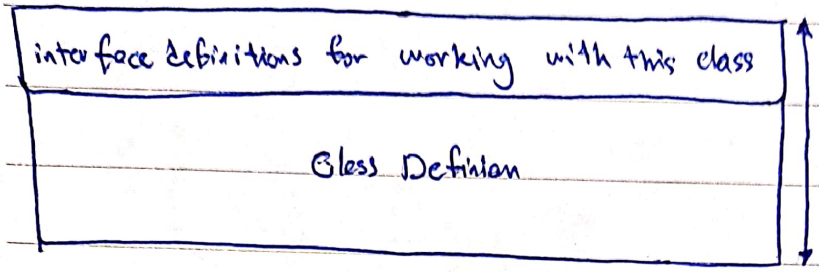
20

User: name, location, Company: name, location, elevatorPhrase, CustomMap: addMarker(), we restricted: access to googleMap, duplicate Definition → first approach:

25

many different types, bad thing: set up dep: btw customMap and all different things: wanted to be on map → best sol: instead: customMap: implements, other classes: implement,

big feature of ts: interplay btw: classes & interfaces, how we are gonna: code,



class definitions: contains methods, to work with class, if given method: hes to recieve: som obj: work correctly, we are going to: specify interface, very low coupling, help ts: show up errors: optionally: use implements, ts: look, will ensure: we implement properly, thats it, three important things: 1. restric api service over, 2. use interfaces to manage dependencies, 3. find errors: correct section: implements clause,

Subject

Year: Month: Date:

5

10

15

20

25

Eiffel

08 Building Functionality with Clases

01

We want to talk about classes in Typescript, classes are blueprint to create different objects, which have fields (values) and functions; we want to consider: First, how classes work and consider their functions, Second; we want to add fields to the classes. we create classes.ts file, then inside it, we create: class Vehicle { drive(): void { log(...)}} ; this is a blueprint; we do not run this method directly. we make instance of a class, then call the function on it: const vehicle = new Vehicle(); vehicle.drive(); this class also have: honk(): void { log(...)}. TS classes are little different from old JS classes.

02

We want to look at inheritance system, we make another class called: Car. anything vehicle can do, Car can also do, instead of copy/pasting all methods,

we write: extends vehicle; now car contain all vehicle methods. we can write:

const car = new Car(); car.drive(), car.honk(). we can optionally override parent

methods, inside Car we can write: drive(): void { log('vroom') }; now the drive

method has been overwriten. vehicle in this example is ~~parent~~ or super class, base class

03

Difference between ES2015 and Typescript is: in classes, in Typescript we have

modifiers: public, protected, private they restrict access, the default is public,

if you don't add any modifiers, modifier will be public, public means it can

be called any time, any where. private means you can only access in the exact same

class. if we make drive() method in class car private, then: const car = new Car()

=> car.drive() will give us error. we can make in class Car: startDrivingProcess(): void {

this.drive(); this.honk() }. now we can call: car.startDrivingProcess(); now we get

another Error: drive() in: Vehicle is public, but drive(): Car is private; so this gives us: can not change modifier at child. note that private only restricts access of functions or variables, it does not add any application security layer. This is the case: we have some methods, that don't want other developers to see it. protected is just like private, but differs in access: child classes also can functions. If we make honk in Vehicle protected, it only can be accessed in Car.

```
class Vehicle {
    drive(): void { log('chugga chugga')
protected honk(): void { log('beep!') }
}

class Car extends Vehicle {
    drive(): void {
        log('vroom') }
}
```

super.drive(): 'chugga chugga', this.drive: 'vroom', this.honk() = super.honk(): 'beep'.

04

now we want to see adding fields to the classes, we write name of property

then annotation => color: string; But we may get not initialized error,

we can write: color: string = 'red'; but it is nicer to specify color

when creating the object: const car = new Car('red'); for this we have to

define constructor. constructor(color: string) {this.color = color}; we either

initialize on same line or inside constructor: color string; constructor(color: string)

{this.color = color}, This is a repeated snippet and has a shortcut: constructor-

(public color: string): we use this constructor shortcut a lot, although the body

of constructor is empty; we should put it.

05

we know how fields work in class, how about inheritance? Car extends Vehicle

=> constructor: provide color string: then error goes away. Car has no constructor,

constructor in parent is called automatically. if in Car we have: constructor(

public wheels: number){} ; then we get Error, it must contain super call, maybe we put: super('red'); but we dont want to hard code: constructor(public wheels: number, color: string){ super(color)}, note we didn't put public before color, These explanations ~~was~~ were very quick, but we will back to them in future.

06

So why do we care about classes? and when to use them? interfaces & classes are primary tool in typescript, different classes work together with help of interfaces.

2022/12/21

09 Design Patterns with Typescript

01

we said we spend a lot of time, on syntax of typescript, now we are finished. we want to work on our first application, we use TS features to write really

reusable code. So our project is as follows: Randomly Generate! user ; then Company show them on the mep. Every user/company, has a location, By doing this project we get more familier with classes. We use parcel-bundler, that is a ts executor inside browser. sudo npm install -g parcel-bundler.

10

02,03

we make a meps directory. then install parcel-bundler, and with parcel index.html we can put ts file in html teg, it will compile ts file to j's and replace the teg. So we have, meps >> src dir we create index.ts inside src dir and run the server by: parcel index.html.

20

Our project structure is as follows: we have a collection of different files, each file is a class, and each class represents one thing. Now based on purpose of 25 our app, our classes are mostly likely are: User, Company and Map. So we create

index.ts, User.ts, Company.ts. if file primary purpose is create & export a class, we write it with upper case letter. but index.ts is not going to export any class.

5

04

now we want to geneveate random data, User properties are user name, location which is a object that contains lat & lng, we are going to Randomly generate theses properties and initialize them inside constructor. we use faker package to generate fake data. faker is going to take care of all data in our app, to access a module that we have just installed if we write: import faker from 'faker' we get error: Could not find declaration of module faker.

25

05

we want to see, what type definitions are, we installed faker module, but we

sew an Error msg. Typescript uses JS libraries; we can import JS codes you and I write to typescript, but ts wants to know types, all code we wrote in JS, has no Type. To solve this problem, we use type definition files, you note that when we installed axios, we didn't specify type definition files. we have to instal typedef file manualy, if it is not installed. For installing type def file we use this: npm instal @types/<library name> => @types/faker They are in Definitely Typed, these files are extremely small, vast majority of time, you have to instal type definition file.

06

If we hover on faker and press Ctrl key, it becomes clickable link, if you click, it will go to: index.d.ts; .d.ts is extenstion for type definitions files, if you see carefully, you will see latitude and longtitude are of

type string. In user class, doing some initialization, in user object we should note that: location object is not initialized, so we can not tell: location.lng = 20, this leads to: can not read property of undefined. So to initialize lat & lng we should initialize location before: location = { lat: parseFloat(faker.address.latitude), ...

07

we want to see what does export statement do in TS. we completed our user, user.ts file contains a single class and we are not going to write actual code to use the class inside this file, instead we export it: export class User{...}

then inside index.ts file, we import it and use it: import {User} from './user.ts'

when exporting, we just use export keyword, So we can export several files, and when importing it, we use {} and we can safely import several things.

if we use : export default 'red' now we can write : import red from './user'

or : import color from './user' : this is not usually used in typescript and

we don't use default exports. So always put curly braces.

2022/12/22

08

Repeating the same process, we create Company.ts, that has: companyName: string catchPhrase: string; and location with lat & lng, we initialize them with faker.

we also note again that: faker.address.latitude() : string So we convert it with the parseFloat() function.

09

Now we generate a new google dev project, then enable support for google maps; then generate api key and then add it to script of our index.html file. We got our api key from: console.cloud.google.com ; we should replace

our key in: key=() ; to check everything is ok, we go to network section,

then to: JS then check if status code is 200.

Inside browser (~~you~~ when you run: parcel index.html), in the console write:

google ; then it should show: {maps: {...}}, We usually instal dependencies, using npm, now ~~we do~~ we have done it, directly inside html file through script,

Now we have a global google variable, without any imports. But if you type google, you get an Error, not found. TS does not understand this variable, so you should use

type definition files. They help TS to understand how third party libraries work

type definition files are not only for npm modules, but also for script tags

added in directly to html file. we instal: @types/google.maps; this is going

to tell TS that, there is a global variable, called google, and all properties

8 functions it has. if you hover on google, you see its type is: namespace

11

We want a map instance: new google.maps.Map() : Err; wants two arguments: Element and options, we see: opt? :> this means opt arg is optional. We provide element by: document.getElementById(id); type of second arg is: MapOptions that is for different options to customize map. we said that: ?: question mark means optional property. we puts {zoom:1}, also center property must satisfy LatLng Literal; we ctrl+click on it and it says that it should have: {lat: number; lng: number}

20

12

we want to hide functionality. Our focus is on: index.ts; Company reference have these properties: companyName, catchphrase, lat and lng. For user: name, lat, lng.

Subject Year: Month: Date:

~~script tags added in directory to have the~~ google Maps: setZoom, setCenter, setStreetView, getHeading, ..., these are methods we expose in our index.ts. other engineers can call these methods and break up our application. Ideal thing is just to expose for map: addMarker(); and hide every other dangerous function from googleMap; So we can just create customMap and call addMarker() on it

13

why to use private modifier? we use it to capsulate google map and expose 15 a minimum amount of functionality we create CustomMap.ts; then export CustomMap so we hide existance of google map; we pretend other engineers can not see inside of this file. private googleMaps: google.maps.Map; now we create user to access its properties and same for company. And for customMap, we use: addMarker and completely eliminate need to global veriable google; because it is wrapped in CustomMap.

14

We want to write code for adding markers, first we write bad code, to implement addMarker; Then refactor into a better approach; addUserMarker(), addCompanyMarker(), we import User & Company types to CustomMap, and create two methods to add marker. This is a bad code, why? in just a little bit. For adding a marker for user we have: new google.map.Marker({map:..., position:{lat: user.location.lat, lng: user.location.lng}})

15

15

So we completed addCompanyMarker & addUserMarker; they look so similar, so what is bad here? These two methods, have a ton of duplication, that will be good for our class to work, as long as you pass smt that has lat & lng.

16

There is a possible solution: approach #1: addMarker(mappable: User|

Company): So what does or(|) do? By using or operator, It takes a look at two types, you can reference a property, if both user & company have that property. So it is limiting properties. There is still a downside, what happens in future if we want to add: Carlot or Park classes to be displayed on the map? we should import them, then write: User | Company | Carlot | Park and ..., so it's a lot of adding and is not a great approach; because it is not scalable and has very tight coupling between our classes.

17

Now we want to restrict access with interfaces, we want to focus on best solution,

Our prev solution of custom map, depend on ~~Map~~ User, Company, CarPlot, Park.

Rather than customMap depending on classes we say: Hey user & Company, you have to satisfy Map requirements; and this is inverting dependency. If you

want to have marker, you should have: the location : {lat: number; lng: number}

property. So we make use of interfaces to invert dependency. Rather than

CustomMap depending on other classes, other classes depend on interface of

CustomMap. interface Mappable

mappable: gatekeeper

addMarker method

User

must satisfy Mappable.

Company

18

Now in customMap we have no direct reference to User or Company; now in

CustomMap we have: addMarker(user) or addMarker(company) and TS does no complain

and checks behind the scenes and inspecting what type user is and because

user & company have location: {lat, lng} there is no Error. Note that there

is no obligation to use implements (opposite of java). We never said inside

user, that User should be mappable. TS did it automatically.

19

we want to show a popup window, we want, when clicked on marker, a popup come. we implement popup with InfoWindow, and then add event listener to it.

adding a popup is a little complicated so we see the documentation. InfoWindow wants a content to be displayed. we put static text: 'Hi there!' for content.

20

we want to customize content inside ~~add~~ window, we want it to be user class type, so it will differ for user & company. we want Mappable to be responsible for this content. So adding new requirement for Mappable: markerContent(): string

Now we should implement it in User & Company. Content can be html also.

21

Our app is now feature complete. if we add color property inside Mappable, we get Err: in idents. Entire point of ts is to help us find errors, and not

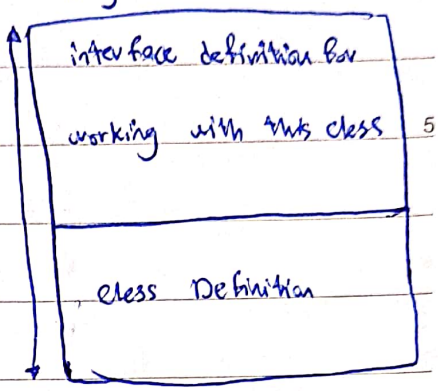
of errors. But index.ts is not the best place to see our err msg. Seeing err in User & Company is a better choice. So inside CustomMap, we export Reportable interface, then in User: class User implements Mappable => by adding implements, we saying ts, help us make sure User implemented Reportable correctly. Now we still see Err msg inside index.ts, but Also see it in User.ts

Note that: implements is not required at en, th

22

we want to review the concepts we learned in this chapter. if other engineers open up: index.ts they see: User: name, location, Company: name, location, catchPhrase, CustomMap: addMarker(); we restricted access to google Map. our first approach ended in many duplication of code, second approach ended in many different types, and the best thing was: setting up dependency between CustomMap and all different

things that wanted to be on map. best solution: instead of custom Map implementing other classes; do vice versa. Big feature of TS is interplay between classes & interfaces. This is how we are gonna code; TS like class definitions, contains methods to work with class, if given method has to recieve some object, to work correctly, we are going to specify interface; that leads to very low coupling. For helping TS to show up errors, we optionally use: implements; then TS looks and will ensure we implement properly. we learned 3 important things: 1. restrict api service area, 2. use interfaces to manage dependencies, 3. find errors, in correct location with help of implements clause.



Subject

Year: Month: Date:

5

10

15

20

25

Eiffel

Subject | Year: 2022 Month: 12 Date: 22

10 More on Design Petterns

01 app overview

arr of numbers: [10,5,18,-3] → [-3,5,10,18], string: 'paJB' → 'aBJoP', Linked List: 10→-3→27→5 : -3→5→10→27, takeCollection: sort it, write out: sorting algorithm : one time → reuse

02 configuring ts compiler

instead of: using: parcel, using: typescript compiler, mkdir src, create index.ts: log('Hi there'), tsc index.ts => index.js: console.log('Hi there'), create src dir, create build dir, src: typescript files, dest: compiled code, js code, configure: ts compiler, tsconfig file: tsc --init → create tsconfig.json, every time: using tsc: checks for: tsconfig: customized setting, to put { ts → src, js → build } : outdir, rootdir: relative path, to where all codes: can be find, outDir: where all compiled go, run: tsc => i need

to compile : all code: from rootDir, compile : into : outDir , tsc -w : watch : all files : inside root Div, if : anything : changed: automatically compile again, =>

Starting Compilation in watch mode, if changing smt: file change detected

03 concurrent compilation and execution

we've got : automatic process : compiling our code, for run : node build/index.js,

automate : run, generate : package.json : npm init -y , npm i nodemon concurrently

nodemon: run our project, concurrently : run multiple scripts, in script section: "start: build": "tsc -w", "start:run": "nodemon build/index.js", "start": "concurrently npm:start:*" => tell concurrently: look at npm scripts, find: every that starts with: "start:", run them all, now : npm start : compile our code, rerun it : every time : a change occurs,

04 a simple sorting algorithm

pretty solid; build setup => putted together, write sorting algorithm, Bubble Sort, double nested for loop, iterate on our array, if element: left > right: swap, explaining steps

05 sorter scaffolding

10

indents: Sorter class: take collection of data, sort it, field: collection: array of numbers,

const sorter = new Sorter([10, 3, -5, 0]), sort method: sort(): void,

15

06 sorting implimentation

actual sort program, const length = this.collection.length; const {length} = this. 20

collection; declaring: new variable length: its type & value: coming from: collection,

hover over: it is number, we got: initial imp: put together, 25

07 two huge Issues

only works: err of numbers, but: we want: string, linked list, what to change, if:

collection : string, numbers: number[] = [1,2,3], numbers[0] // 1, numbers[1] = 3 =>

[1,3,3] ; strings: const color = 'red'; color[0] // 'r' but color[0] = 'y' // color: 'red',

strings: immutable, "Xaaa" => sorted we expect: "aaaX", in JS: "X" > "a": false,

charCodeAt => "X": 88, "a": 97, because of char code, so: comparison: > : numbers ✓

strings X, this.collection[j] = this.collection[j+1] : numbers ✓, strings X, different

types of collections: different comparisons different swapping,

15

08 typescript is really smart

right now: numbers & strings, first: bad solution, // If collection: number => do

20

this, // If collection: string => do this, number[] | string : restrict props:

on both: common, working on: array, direct access element, for string, direct access:

25

reading, not writing, index signature in type 'string' only permits reading,

Subject | Year: Month: Date:

09 type guards

result of: number[] | string, type guard: type check, clarify: type of value,

if: this.collection instanceof Array => inside if: this.collection. => props of arr,

TS: way smarter, you would expect, same thing: string, syntax: different,

if: typeof this.collection == 'string' => inside if: this.collection.: see all methods

associated with: string, quick note: type guard: primitive: typeof: number,

string, boolean, symbol, typeof []: object, instanceof: every other value, created

with: constructor, Sorter, Date, ..., Array,

20

10 why is this bad?

collection: number[] | string | LinkedList ..., adding if statements in sort function,

25

and implementation: 15 sortable things, 15 if: doing almost same thing

11 entracting key logic

little bit of, intermediate: refactor, two operations: going to be, customized, based on: data structure, comperison & swapping, entrect comparison: helper function, another class, Sorter.ts: sort(): { for() for() if (compare) { swap? }, NumberCollection: custom swap & compare: for numbers, Sorter, %100 generic, Sorter: obstreect data

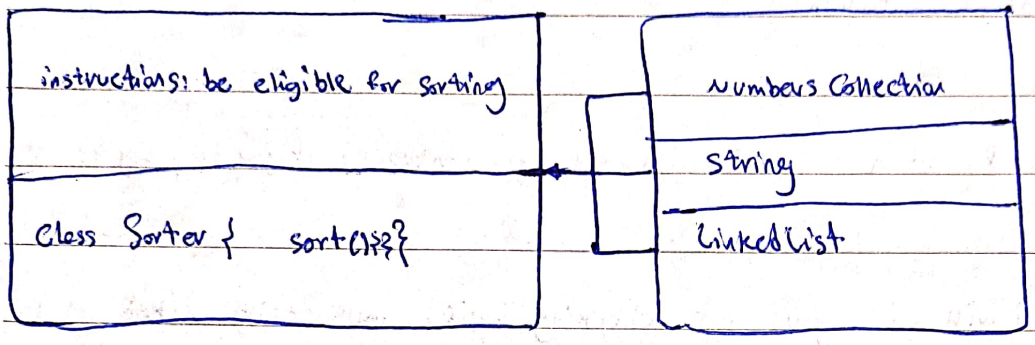
12 separating swapping and comparison

make: NumbersCollection.ts => data: numbers, implement compare & swap, we should: make sure: there is length field; length(): number { return this.data.length }

another way: natural: property not function: getter or accessor, get length(): number { return this.collection.length }, const c = new NumberCollection([1,2,3]), c.length, point of get, reference length, as field, in Sorter: import NumbersCollection, use NumbersCollection: methods: at sort,

13 the big reveal

fix: Sorter: only NumbersCollection, but we want more,



instructions: length, swap, compare

14 interface definition

make: interface: Sortable: in Sorter class, name of args: btw interface and class

which implements it: do not have to: match up, only reason: same: match up:

give them: context, vague idea of meaning,

15 sorting arbitrary collections

creating: CharactersCollection.ts, comparison: using toLowerCase(), implementation

Subject Year: Month: Date:

of: length, compare, swap; converting: char array, swap, convert to string,

16 linked list implimentation

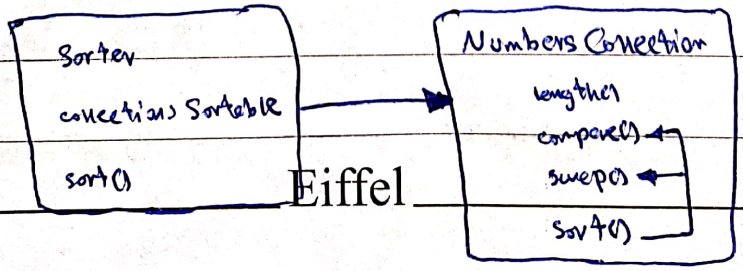
interfaces: useful: set up a type, also: set up: contract: if you impx y z: all functionality, i will give you, not checking logic, implimenting linkedlist logic, for: comparison, length, swap

17 just one more fix

now: create NumbersCollection, create Sorter, call sorter.sort(), then print, tabourious, boring, better: numberCollection.sort(),

18 integrating the sort method

how to add, sort method, our three classes: number, string, linkedList=> inheritance, now:



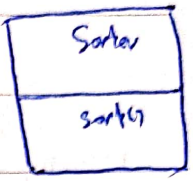
: refactor :

Eiffel

Subject | Year: Month: Date:

19 issues with inheritance

refactor: Sorter: sort() : parent of, NumbersCollection, ..., LinkedList,



in: NumbersCollection: extens Sorter, in constructor: super(), const {length} = 5

this.collection; => const {length} = this, we are: never going: have instace

Sorter: directly, we just want: take sort method: copy it to child classes,

20 abstract classes

using: this.swap(), this.compare() => Err, because: we want: subclasses to:

implement them, Typescript Expectation:

Sorter	NumbersCollection
✓ sort()	✓ length()
X swap()	✓ compare()
X compare()	✓ swap()
X length()	✓ sort()

in reality: we implement methods: in NumberCollection

not: Sorter, we need to: make sure: Typescript: finds out: reality; Abstract

Class: create obj directly X, only: as parent class, methods: Can contain: real imp

: such as: sort, implemented methods: refer: methods that has not imp,

child classes: promise: imp: not imp methods,

21 why use abstract classes

Sorter.ts: export abstract class Sorter {...}, tell ts: we will, eventually:

define: length(), compare(), swap(): abstract length: number; ...

we will treat: length: as property: field,

22 solving an issues with abstract classes

const numbersCollection = new NumbersCollection([-1, 3, 7, 5]); numbersCollection.sort()

we should call: super(): inside constructor: otherwise: get error, if child:

does not define: constructor: Sorter constructor: automatically called, if define:

should call super, by using abstract class, you may think: we dont need

interface, interfaces: still super key in TS

023 interfaces vs abstract classes

we don't need interface anymore, quick compare & contrast: interface: set up contract btw classes, different objs want work together, loose coupling

abstract class/inheritance: set up contract different classes, strongly couples classes together, Sorter without childs childs without Sorter pointless,

Subject

Year: Month: Date:

5

10

15

20

25

Eiffel

Subject Year: 2022 Month: 12 Date: 25

10 More on Design Patterns

01

We want to impliment a generic sorting algorithm, For example: array of numbers: [10,5,18,-3] → [-3,5,10,18], string: 'poaJB' → 'aBJOP', Linked List: 10 → -3 → 27 → 5 => -3 → 5 → 10 → 27. So we want to take a collection and sort it, and we want to write algorithm one time, and reuse it.

02

Instead of using parcel we want to configure and use typescript compiler, So: mkdir sort; cd sort; touch index.ts, we write: log('Hi there') inside it. Now if we compile this file with: tsc index.ts; we get: index.js inside sort folder; we create src and build directories, we want src to have all typescript files and dest to contain all compiled JS code. We configure ts compiler by changing tsconfig.json file. we create this file by: tsc --init. To put: ts: src / js: build we change:

outDir & srcDir : is relative path to where all codes can be find. outDir is where all compiled codes go. Now if we run: tsc : it will find all code in rootDir and compile them to outDir. and if we run: tsc -w : it will run on watch mode and watch all files inside rootDir to see if any files changed, and it will compile it automatically again.

03

we have got automatic process for compiling our code, we can run our program by: node build/index.ts, but we want to automate the run process. for this purpose we have to install new packages, so we generate package.json file by: npm init -y

so now package.json is responsible for our dependencies. npm install nodemon concurrently. nodemon is for running our project every time js files changed. and concurrently is for running multiple scripts. now in script section, in package.json:

"start:build": "tsc -w", "start:run": "nodemon build/index.js", "start": "concurrently "npm:start:*"". The last instruction tells concurrently to look at npm scripts, then find every instruction that starts with: "start" and run them all, now: npm start: compiles our code and reruns it every time a change occurs.

10

04, 05, 06

we have a pretty ~~soli~~ solid build setup putted together, we want to write Bubble Sort algorithm, it has two nested for loops, and iterates on our array and if element of left is greater than right, it swaps them. In index.ts, our sorter class takes collection of data, and sorts it. First to be simple, we have collection field, that is an array of numbers; const sorter = new Sorter([10,3,-5,0]) and we have a sort method.

To implement actual sort program, we can say: const length = this.collection.length or we can write: const ${length} = this.collection => this declares new variable length

and sets its type and value comming from the collection.length, if you hover over it, its type is number.

5

07

Now we here some issues, Our sorter only works with array of numbers, but it does not work with strings or LinkedList, now what to change in our code, to work with strings? we should note that: numbers = [1,2,3] : numbers[0] // 1 numbers-[1] // 2, and for strings we have: const color = 'red' => color[0] // 'r' 15

but there is a difference between array and strings: numbs[0] = -1 => // numbers: [-1,2,3]

color[0] = 'y' // Color : 'red' => So strings are immutable and also: "Aaaa" 20

the sorting we expect is: "aaaX" but in JavaScript 'X' comes before 'a',

because: char code of 'X' is 88 and char code of 'a' is 97, So comparison works correctly with numbers but does not work correctly with strings & and also

because of immutability, we can not do this: this.collection[j] = this.collection[j+1]

for strings. So for different types of collections we need different comparisons

and swapping.

08

Typescript is really smart, Right now we want to support strings & numbers,

first we want to see bad solution, => If collection: number do this... else If

collection: string do this ..., So we use number[] | string: this restricts properties

on properties which are both in numbers[] and string. For array, we have direct

access for reading and writing both in string, we have it just for reading. If we try

to write: color[0] = 'b': Index signature in type string only permits reading.

09

Result of: number[] | string, we have a type guard, for type checking to clarify

type of value. if : this.collection instanceof Array ; inside if this.collection, has properties of arr. TS is way smarter than you would expect, this is same for strings, But synten is different, if: typeof this.collection == 'string' : inside if statement, if we write this.collection. we see all methods associated with string. Quick note: type guard for primitive types are : typeof and primitive types are: number, string, boolean, symbol. you can use typeof with for example [] but: typeof [] : object. we use instanceof with every other primitive value, created with constructor: Sorter, Date, Array, ...

So why this code is bad? after a time, if we add more classes: collection: number[] | string | LinkedList ... and we should add if statement in our sort function, and implimentation for it. Suppose we have 15 sortable thing, then we have 15

if statements, doing almost the same thing.

11

with a little bit of intermediate refactor, these two operations: swap & compare are going to be customized, based on the data structure. we want to extract comparison, into helper function, an another class: Sorter.ts which has a sort() method that is in the form: { for () { for () { if (compare()) swap() }} . NumbersCollection will contain custom sort for numbers, CharactersCollection, will contain that for strings.

This sorter is 100% generic.

20

12

We make NumbersCollection.ts which has a data of number[], we implement compare and swap, we should make sure there is a length field, we can define it as function; length() { return this.data.length }, But there is another way, that is: getter or

accessor function: get length(): number { return this.collection.length }, Then we

can write: const c = new NumberCollection([1,2,3]); log(c.length), So the

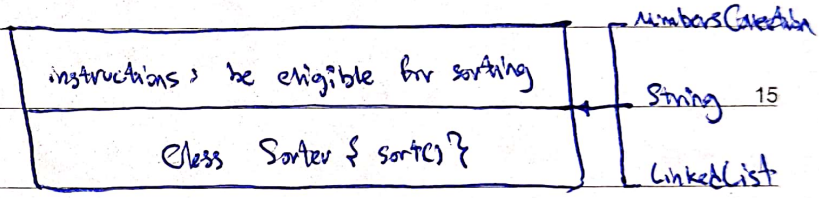
point of get is to reference length as field, in Sorter we import

NumberCollection and use its methods at sort.

13

now we want to fix Sorter, because currently it only works with NumbersCollection

but we want to have:



instructions are to implement: length, swap, compare

14

we want to define interface Sortable in Sorter class, it is a contract between

our classes to use Sortable class. Note that the name of args between interface and

class which implement it, do not have matchup, the only reason for them to be

seme is giving conten and wege idea of meaning.

Subject

Year: Month: Date:

5

10

15

20

25

Eiffel

11 Reusable Code

01 project overview

spread sheet, football match, CSV data: load → parse → analyze → Report,

node std lib

code we write

02 project setup

in the past: package.json: scripts: "start:build": "tsc -w", "start:run": "nodemon build/index.js", "start": "concurrently npm:start:*", builds run simultaneously,

mkdir starts, npm init -y, tsc --init, npm install nodemon concurrently, create src and build directory, change: tsconfig.json: rootDir and outDir, / root: ./src / out: ./build

first time: see: Error: nodemon: try run: before: being compiled

03 type definition files again

How to: open up & read file, www.nodejs.org: FileSystem, fs.readFileSync: give

path, gives content as string, import fs from 'fs', in node, just: require,

in TS, cannot find module fs, we have to install type definition file, Some

for: node standard library, fs, http, os, usually: npm install @types/{name of library}?

So, you may think: npm i @types/fs, But no!, for node standard libraries:

same file: npm install @types/node

of reading csv files

now: we have fs module, read csv file, read file sync: options: encoding: utf8

big string: parse: useful info, string —split('\n')→ lines: string[] —map+split→ string[][]

inner array: one row data, .map((row: string): string[] => { return row.split(','){,

each inner array: one football match => date, hom team, away team, ... : home team

goals: number, but what we have: string, convert: number, we do it: later

05 running an analysis

now : data : parsed, we want : one type, Man United : # time, won game,

every match, man united, home & won or man united : away team & wins, for : lets

match of matches, 0 : date, 1 : hometeem, 2 : away team, ..., (match[1] == 'man united'

&& match[5] == 'H') || (match[2] == 'Man United' && match[5] == 'A')

06 losing dataset content

couple of issues : go through : refactor : features of TS, magic string comparisons;

match[5] === 'A', other engineers will not understand, homewin = 'H' => if match[5] ==

homeWin =, thats better, we wrote : homewins every win ! but : there is : draw,

to fix : const draw = 'D' => another Issue : draw never used, another engineer !

come along : delete it! next video : solution for this

Subject | Year: | Month: | Date:

07 using enums

replace all the identifiers: homewin, awaywin, draw => const MatchResults:

possible outcomes of match: { HomeWin: 'H', AweyWin: 'A', Draw: 'D' } other engineers: look & say:

why this: object? shows smt important? instead: we use: enum

enum: hold numbers or strings, enum MatchResult { HomeWin = 'H', AwayWin = 'A', Draw = 'D' }, why use: enum over object? signalling: other engineers:

collection of closely related values, type: MatchResult. HomeWin: MatchResult

08 when to use enums

20

quick note about: what they are, when to use, syntax: like normal objects,

log (MatchResult. Homewin): 'H', possible: enum MatchResult { HomeWin, AwayWin,

25

Draw }, in reality: when ts compiles → js: enum: object, primary goal: signal

to: other engineers, purpose of your code, whenever: small set of values:

Eiffel closely related, known

at: compile time

any use enum: when: know: all values: at compile time, examples of when to use/not use enums,

09 entracting csv reading

we had some issues: src of data: hardcoded, imegine: we tried: get football data: from api, if change: src of data: should: change: hab of our current ade, create: CsvFileReader class: fields: filename, data: string[][], Methods: read(): void, read method: responsible for: opening file, persing info, now: const reader = new CsvFileReader('football.csv'); reader.read(); if: get info: from api: just comment these two lines,

10 data types

Back to issues we had: our dateevr: on strings although we have: date, number

Subject | Year: | Month: | Date:

10/08/2018, Man United, Leicester, '2', '1', H, A Merriar

parseDate | parseInt | parseInt | matchResult | do parsing, update types

Date | string | string | number | number | string | string

5

// converting date strings to date

'10/08/2018' → Date object, how we create Date object: JS: new Date(2009, 0, 15),

10

split string → extract month, day, year, → subtract month by one → feed to Date,

we can export functions same as classes, months inside Date, zero indexed, January:

15

0, So, pieces[1]-1, we use map convert them one string → number,

// convert row values

20

add another map when reading csv, inside read method, inside map for each

row (match), return [parseDate(row[1]), ~, parseInt(row[2]), ~], what about 25

matchResult?

Subject | Year: Month: Date:

13 type assertions

In the last video: we converted: Date and number, what about: MatchResult?

we need: MatchResult: inside: CsvReader file: export it from: index.ts X: 5

we do not: export: from index.ts, row[5]: tell TS: this: going to be: one of the

results of: MatchResult, row[5] as MatchResult: type assertion, overrides

TS default behavior, problem: in the second map (in read method): return type:

any, what to do? 15

14 describing a row with a tuple

20

Now: ~~OK~~ we have: converted: all: correct type, because we return array:

(Date | string | number | MatchResult)[], we said why we: use: objects over tuple

25

before, [brown, true, 40], { color: brown, boolean: true sugar: 40 }, we want: use: tuple: over array

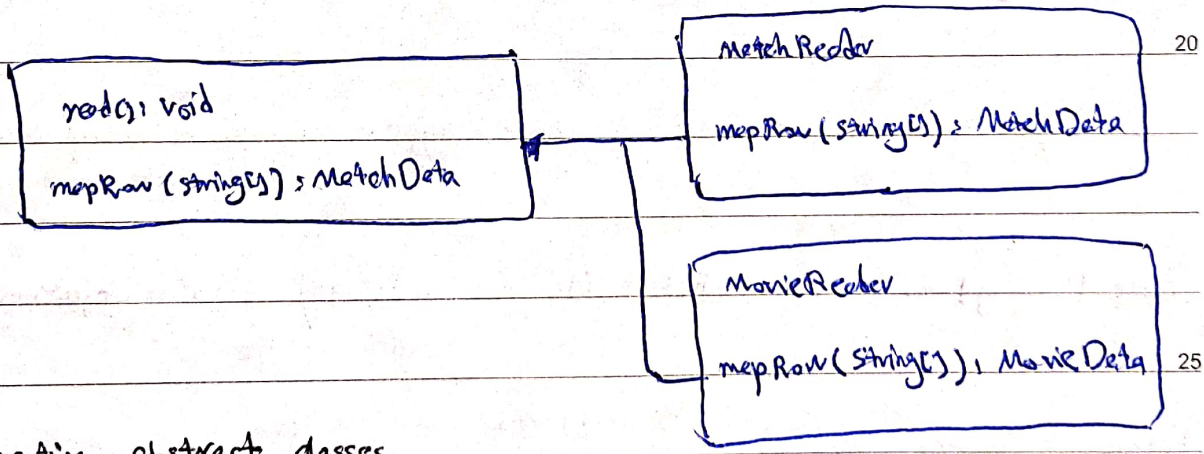
type MatchData = [Date, string, string, number, number, MatchResult, string]

15 not done with filereader yet

why: we created CsvReader.ts: 1. reusable src of info, ib change to api: delete a lot of code 2. can use on: future projects, But: is it: reusable now? no, a lot of: reference: match data, we went: refactor: make it: reusable, cp CsvFileReader.ts CsvFileReader.bak

16 understanding refactor 1

first part: first map: in read method: generic, second map: custom logic: put logic second map: helper function: mapRow, turn CsvFileReader: abstract class:



17 creating abstract classes

create: MatchReader.ts: implement: mapRow, mark CsvFileReader: as: abstract

abstract mapRow(row: string[]): MatchData => MatchData : prevents: being reusable

we could: cheat: put: any: not a good idea, discuss it: next video

18 variable types with generics

we don't want: reference: to: MatchData in: CsvFileReader: because: its/:/ss

upon: football.csv, could change it to: any: not good idea: avoid any, smt called:

generics, Generics: like: function arguments, when passing: arg→ function:

customize body behavior, generics: define: classes/ functions: customize different types

: inside classes/functions, allow: define: type: property/arg/return value: future point,

used heavily: reusable code, const addOne = (a:number): number {return a+1},

same for: addTwo, addThree, .. duplicating function, much better way: add

rather than: hardcode: get arg b: add(a:number, b:number): number {return a+b}

imagine: when: add(a,1) => 1 is replaced with b, make use: arg: customize

function behavior, with that in mind; class HoldNumber {data: number}, class HoldString {data: string}, ... => just like functions: duplicateX : for one field, instead: using generics: class HoldAnything<T> { data: T }, now: const holdNumber = new HoldAnything<number>(2), same for string, reuse: as many type as you want: any type, now we can: replace: MatchData => generics: T, why T? just convention, shorten of type, but: we can name: anything we want,

19 applying a type to a generic class

we could use: generics: for anything: inside class, use for: add(a: T): T {...}

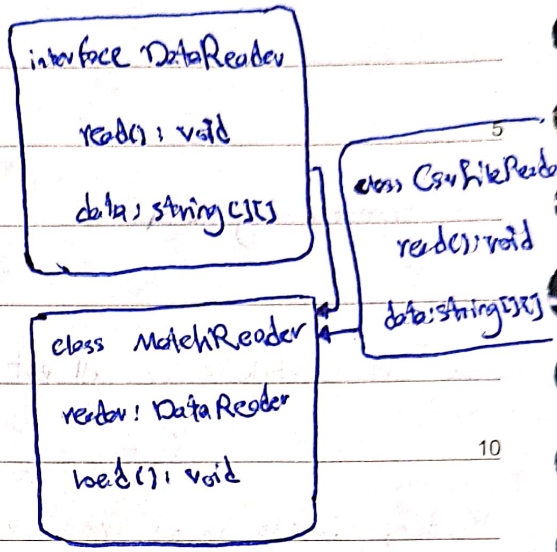
so: we put: MatchData -> MatchReader.ts: matchReader extends CsvFileReader<T>

This was: our MatchReader: refactor #1: finished, used: AbstractClass

20 alternate refactor

we achieved: code reuse: by: AbstractClass + Generics, pretty tricky: imp

although its reusable; its hard to read, find out; what's going on, move our CsvFileReader & MatchReader, new dir; inheritance, mv: CsvFileReader.ts. bak → remove bak, #2 approach: heavily: based on: interfaces,



to be a: DataReader; you should: have read function & data field, CsvFileReader; is a DataReader; that has: implemented: DataReader, we may have: several: readers: CsvFileReader, ApiReader, JsonReader, ...,

21 interface based approach

heavily: upon: interfaces: rather than: inheritance, create: MatchReader.ts: inside it: create: class MatchReader, & interface: DataReader,

22 extracting match references again

start refactoring: CsvFileReader, CsvFileReader implements DataReader, removing

all MatchData references from CsvFileReader —move→ MatchReader, ensure; match reader; data read, and converted,

23 transforming data

we got: CsvFileReader: all put together, last thing to do: imp load method: match-Reader, inside load: reader.read(), then: ~~data~~ matches = data.map(~) // convert to correct type → new property in class

now: our: MatchReader & CsvFileReader: ready,

24 updating reader references

last thing: update index.ts, to create: instance of: MatchReader, we should pass: smth that: imp: DataReader,

```
csvFileReader = new CsvFileReader(path)
matchReader = new MatchReader(csvFile)
```

two separate classes, where match reader

```
matchReader.load();
```

getting: info: on the fly, load method: tells: get all the data & transform it,

now : we have all info : matchReader.matches,

: چیزی که من فکر می‌کردم این بود که من خواستم جوری بنویسم که برنامه‌مون خیلی وابسته به Match نباشه،

یعنی هر فایلی که مثلا csv باشه رو بخونه، ولی این طوری نوشتم که برنامه‌مون جوری باشه که فایل football.csv

رو بخونه اگر هر src ای مثل api، فایل json یا بقیه بخونه

2023/07/07

25 inheritance vs composition

completed: two big refactor, which is better? inheritance vs composition

inheritance: CsvFileReader, child class: MatchReader, Composition: class MatchReader, reference: DataReader, which: CsvFileReader & ApiReader: implement, benefit: swap in: different readers, inheritance: is a, composition: has a, in composition: we deligated: loading op: outside obj,

26 more on inheritance vs composition

modeling window, window & wall: common properties, make superclass Rectangle

: put common: in Rectangle, suppose: circular window, make superclass Circle,

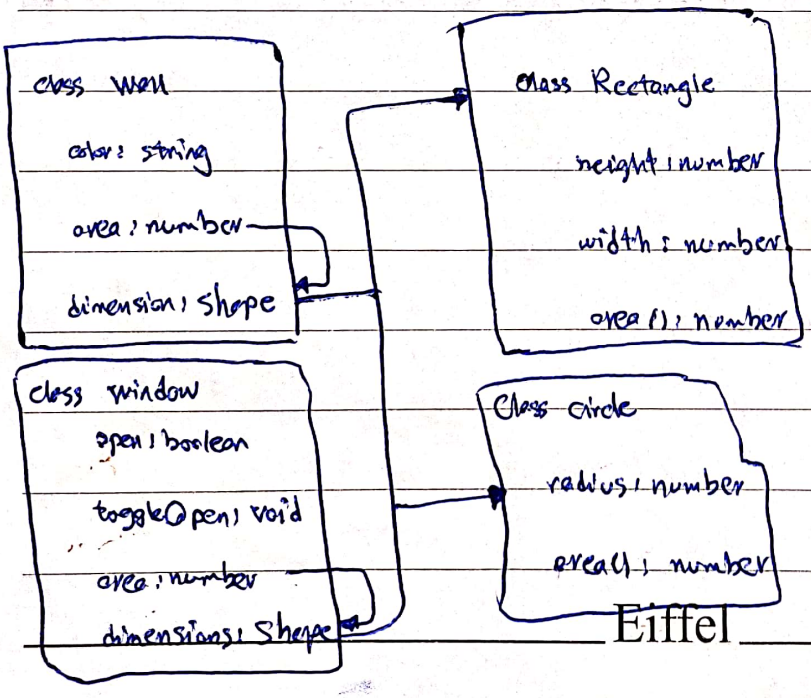
which has: radius & area, we have code duplication: circular window & rectangle

window: just differ: shape, abandon: inheritance: use composition: create

class window & wall: ~~shape~~ => dimension (property), interface shape, Rectangle &

Circle: Shape, easily swap, like our readers,

27 a huge misconception around composition



Favor object composition over class inheritance.

Under Delegation, keeping reference to rectangle, delegate area calculation to it,

Eiffel

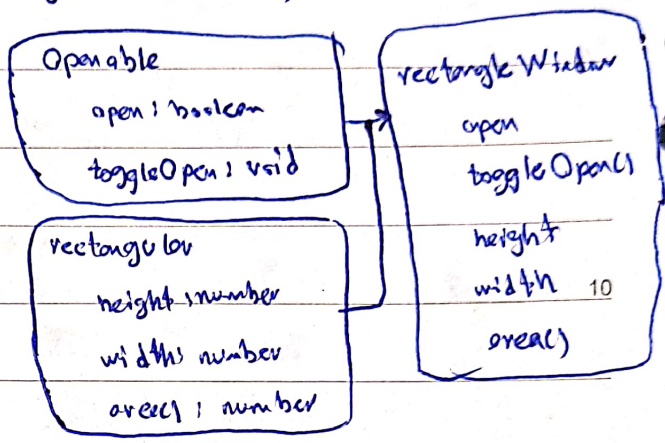
Subject Year: Month: Date:

Object.assign(state, canFight(state)) : copy all attributes of : first → second,

composing object : by its properties, Object.assign(state, rectangular(state),

openable(state)) => state becomes: rectangular & openable, what we are doing:

composition ✗ multiple inheritance ✓



28 goal moving forward

Issues: variable : named : specific team : men united

interface Analyzer

run(matches: MatchData[]) : string

AverageGoalAnalyzer

class summary

WinsAnalyzer

interface OutputTarget

print(report : string) : void

ConsoleReport

HtmlReport

analyzer

outputTarget

buildAndPrintReport => deligate to: analyzer & outputTarget

29 a composition based approach

all files: reference: MatchData: move it, its own file: otherfiles: not depend on:

MatchReader

Eiffel

Subject

Year: Month: Date:

matchData : needs matchResult ; so import it here, 01:25