

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-gridev

02 typescript overview

what is typescript? typescript = js + type system, still writing JS, oddings oddings

syntax, type system: 1. help catch errors → development, 2. usual JS: execute, 2. type

annotation → ts compiler, type annot, little comments, go away active, off during dev →

browser: no idea, at least: get a js file, 1. performance optimization ✕,

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

me: execute TS X, execute JS ✓, typescript lang. or 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 set-up

set our local machine to 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,

of a first app

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

steps: take long: api: used for fetching data, create new proj dir, or package.json,

install axios + make req, write code, take look: api: jsonplaceholder.typicode.com

→ Take JSON api, Axios: 200 todos, no need all items: just first: jsonplace-

holder.typicode/todos/1, mkdir fetchjson, or 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. \Rightarrow axios.get(url), async \Rightarrow return promise,

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

compile \rightarrow plain JS, tsc index.ts \Rightarrow 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) \Rightarrow ugly, extract: id, title, completed, print nice, pull off
const finished = todo.finished

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

when logging them: all: undefined \Rightarrow what's wrong?

07 catching errors with typescript

ID, Title, finished \Rightarrow actual name: id, title, completed, we don't know! this bug:

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

property: id, title, completed \Rightarrow using typescript: interface, interfaces: define

ALVAND structure obj, ignoring: userId \Rightarrow we can do this, respond data as Todo
in

Subject :

Year. Month. Date.

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

08 catching more errors

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

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

now, boolean → string: compile err, we saw two examples: ab: most common errors,

both case: run: find err; typescript: compile 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 +

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

how to write reusable code, our main 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, types easy way, values properties & functions has, array, types object

type, "red": a string, a value: all properties & methods: string has, string: character,

concat(), ... , shortcut: properties & functions: here: is a string, every value: int has

a type,

03 move on types

types: string, number, boolean, null, Date, Todo, not restricted: this types: every

value: has: type, Types: primitive, object types, primitive: number, boolean, null,

undefined, string, symbol, null, object types: any ~~types~~ types: we create, some

for functions, arrays, classes, objects, why divide? think 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 the our code, other engineers: labeling
ALVAND

types ⇒ understand

our code

Subject :

Year. Month. Date.

04 examples of types

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

today => auto popup, tell methods, based on Date, const person = { age: 29 }; =>

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

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

05 where do we use types

we get, Aha, why care, examples, => where we use? everywhere,

2022/12/15

03 Type Annotations in Action

01 type annotations and interfaces

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

we write, what type, is variable, Type Inference: typescript tries to figure out:

ALVAND type, Type Annot \leftrightarrow Type Inference

↳ we tell typescript ↳ typescript guesses

Subject :

Year. Month. Date.

02 annotations with variables

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

number = 5 => we can't 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 litered annotations

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

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

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

methods: 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 nested, 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` \Rightarrow let apples = 5 : both, when hover? apples: number,

05 understanding inference

`const color = 'red';` if dec & init: same line \rightarrow typescript: figure out type,
var Decl var Initialization

if: `let apples; apples = 5` \Rightarrow typescript: can't figure out, apples: any, when? Type

Inference: always! \Rightarrow a lot 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' \Rightarrow boolean, '4' \Rightarrow number`

`'{"value": 5}' \Rightarrow {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: auto 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: envy =>

let foundWord: boolean;

09 when inference does not work

third case? we want a variable: 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 | numbers,

we looked 3 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 about functions before => const logNumbers (i: number) => void =

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

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

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

return a number,

02 inference around functions

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

example: 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, never to add! see annotation, type inference: only about 1 return value / function,

ALVAND

Subject :

Year. Month. Date.

const subtract = (a: number, b: number) => { a - b; }: never return statement, return

value becomes void, so always odd annotation,

03 annotation for anonymous functions

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

some for anonymous functions: const multiply: function(a: number, b: number): number;

arrow function, named function, anonymous function, same,

04 void and never

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

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

never

= (msg: string) => { 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. _____

a5 destructuring with annotations

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

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

ES2015: {date, weather}, destructuring: replace variable name, fields:

forecast → {date, weather}

a6 annotations around objects

until now, variables & functions, but we're talking about object destructuring,
^{want to}

const profile = { name: 'elen', 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 } } = profile, 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

destructuring, functions: λ too rely on annotation, variable as much as can's inference

ALVAND

Subject:

Year. 2022 Month. 12 Date. 15

01 Getting Started with TypeScript

01. Help/Info

This section talks about how to get help, there are several ways to access to Stephen Grider, udemy website is one way, also stephen 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 can still write JS code in TS, but there is additional system.

Type system has some advantages: 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 \Rightarrow TS-Compiler is a command line

ALVAND

Subject :

Year. Month. Date.

tool, that gives us plain JS from TS code, \Rightarrow 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 equivalent of ts code side by

side by its JS. At last, and as summary we can say that ts is js + extra documentation,

and ts has no effect on code; it's just like a friend sitting behind you, while

you code.

03.04

We want 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 installation. If you got permission denied error,

you also should use that command with sudo. That command 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 managing our dependencies. Now we install axios with `npm i axios`, we make get req to `jsonplaceholder.typicode.com/todos/1`, this site contains JSON that you can get. todos contains 200 todo item, we don't 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 sync 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(response.data)`, it is ugly, so we

want to extract `id`, `title` and `completed` and print them prettier. So we pull

`todo`

`data` prop and assign them to another temp variable. \Rightarrow `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, title, 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 = response.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)
{}^{fun?}

Now if we put boolean → string, we get compile time error. We saw two examples

of most common errors, both cases we got 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 JS has a type.

03

Some types are: string, number, boolean, Date, Todo and we are not restricted to

ALVAND these types, every value has a type. Types are: primitive & object types,

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 either, why we care? we declared Bob with : id, title, empId; so

if he's 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 types

today. \Rightarrow something will popup and tell methods based on Date. const person

= {age: 20} \Rightarrow if you hover on person, it shows all info about person. class Color

sq, const red = new Color(); \Rightarrow if you hover on red, it shows its type is Color.

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

ALVAND

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 \Leftrightarrow 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 \Rightarrow that's compile time error. with every types: primitives & objects

We can use types. let speed: string = 'fast' ; let nothing: 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 arr: string[];

this can only contain array of 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 ~~type~~ initialization; also with classes we have:

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

; 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 nested,

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

=> void, = (n: number) => {console.log(n)},
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 \Rightarrow typescript

cannot figure out the type, and apples gets any type. So when we have type inference

with variables? always! \Rightarrow you will see a lot of code: let apples = 5; but rarely

see: let apples: number = 5; But there is three scenarios that you use

implicitly.

06

we want to consider that three scenarios: 1st: function returns 'any' type and

we need to clarify the returned value: let coordinates = JSON.parse(x); now

coordinates gets 'any' type. we expect from JSON.parse that for these inputs,

return that outputs: 'false' \rightarrow boolean, '4' \rightarrow number, if "value": 5 \rightarrow {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 does not 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 2nd type is declaring

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

`words = ['green', 'blue', '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 3rd use, we want a variable type that can not be inferred,

For example suppose 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 (numbers[i]) numAboveZero = numbers[i] => gives true,`

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

at Annotations with Functions & Objects

01

we said we will consider Type Annotations & Type Inference for variable, functions

and objects. we ~~have~~ talked about variables, you may think we talked about

functions also when we said: const logNumber: (i:number) => void = ~~i~~ ^{log} (i:number)

=> {log(i)}[?]. For type inference for functions, typescript tries to figure out

the type function will return, const add: (a,b) => {w?} , if we code this, we

get a warning : type a is 'any' so we should write: (a:number, b:number) number

^{if}

=> although this gets error if we do not return anything in body : non void

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

ALVAND

Subject :

Year. Month. Date.

return type not logic of our function. So if in odd function we return ab that

is ok. There is no type inference for arguments, so we have to always add annotation - type 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 infers void; we conclude that we should

always add return type annotation.

08

Annotation for anonymous functions is some: function divide(a:number, b:number) =>

number => const multiply = function(a:number, b:number): number. This serve

for all of these) arrow function, anonymous function.

09

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 want to return

anything, don't put never: const throwError = (msg: string): string => { if (!msg)
throw new Error(msg); return msg; }

Q5:

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

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

as in ES2015; we can destruct and replace variable name in fields: forecast

→ { date, weather }

Q6

SAY went to consider annotations around ~~functions~~ objects, until now we

checked variables & functions. Let's talk about object destructuring.

Suppose we have: const profile = { name: 'alen', age: 20, coords: { lat: 10, lng: 20 } },

setAge(age: number): this.age = age; ?

[Pause] look of paper

ALVAND

05 Mestring Typed Arrays

01 arrays in typescript

arrays in ts, quick Def: plain JS Array; push, pop, map, foreach, one difference: 5

stick one type, only strings, one specific type in: array, create: arrays.ts,

we put type annotation? when: empty arrays const arrMakers: string[] = []; 16, 10

arrMakers: const arrMakers = []: 'any', for 2d: string[],

15

02 why typed arrays

why care? why separate topic?, some advantages, some disadvantages,

20

ts: knows: arrMakers: arr of string, const car = arrMakers[0] => hover: car: string,

also works: arrMakers.pop(), also: arrMakers.push(100) => Err: incompatible values

25

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

string => { return car?; } or s return car.toUpperCase() => gives auto complete,
Eiffel

03 multiple types in arrays

how to use arrays \Rightarrow multiple types in them, const important Dates = [new Date(), '2022-10-12']

\Rightarrow type: (string | Date)[]

5

04 when to use typed arrays

10

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

see: what differences: tuples & arrays

15

2022/12/18

05 tuples in typescript

20

01 tuples in typescript

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

25

tuples: mix a lot of types, f colors: 'brown', carbonated: true, sugar: 40%, one

object: one drink, represents this object: array: [brown, true, 40] \Rightarrow most

Eiffel

info

in arrays: we should remember, color, carbonated, sugar, differences 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] \Rightarrow (string, boolean | number)[]; we

can swap elements, we want the order: break order \rightarrow data model: breaks out, 10

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

rather than: repeat: [string, boolean, number] \Rightarrow type alias, type Drink = [string,
boolean, number]

but no array ~~exists~~ created, now: const pepsi: Drink = ..., reuse,

we are not using tuples: very often, tuples: not super useful, 20

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,
Eiffel weight: 3354}

object: immediately observable, tuples much harder, what's going on?

2021.7.21.18

03 The all Important Interface

5

01 interfaces

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

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

interface: new custom type,

15

02 long type annotations

20

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

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

long annotation 25

we are going to duplicate this: for each car,

Eiffel

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

15

little note: about: interfaces, not limited: expressing: primitive values, any: type

we want: also have function, interface Vehicle { name:string; date:Date; broken:

20

boolean; summary(): string?; } and objects: should never: all of this,

05 functions in interfaces

25

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

if > interface vehicle { summary(): string?, const make = "nissan elvis",

year: new Date(), broken: true, summary(): string? return 'Hello?'

> old elvis > satisfies > Vehicle interface, Vehicle: only check > to see summary,

any question > get asked: do you have summary?, additional properties: old elvis!

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

ab code reuse with interfaces

15

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

summary(): string? return 'my drink, has this sugar?', drink & old elvis:

20

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

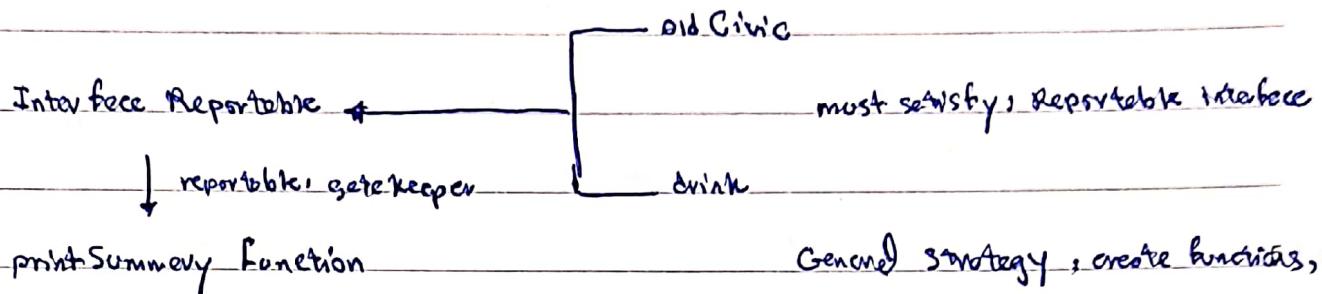
summary (reportable: Reportable), point: single interface, different object, encourage?

25

write generic methods,

Eiffel

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: values implement interface

can work with: function

15

20

25

Subject _____

Year: _____

Month: _____

Date: _____

5

10

15

20

25

Eiffel

05 Mastering Typed Arrays

01

We went to see typed arrays in TS. First we look at their quick definition, ts

5

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

10

arrays.ts. We should put type annotations when declaring empty array; for example

we have: const carMakers: string[] = []; if we write const carMakers = [];

15

Our array gets 'any' type. Also for 2d arrays we should write: string[][]

02

20

so why do we care about typed arrays, that we have created a separate section

for them? they have some advantages and some disadvantages, ts knows that

25

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()

Eiffel

and also writing `carMakers.push(10)` will give us `10`; incompatible values,

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

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

`toUpperCase` => it 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.

Eiffel

06 Tuples in Typescript

07

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

5

inside tuple, we can mix a lot of types. Suppose we have this object: { color: 'brown',

10

carbonated: true, sugar: 40 }, this object represents one drink. We can represent

15

this object as array s ['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.

20

02
So when to use tuples? they are not mostly used, but they are used when working

25

with csv files, but suppose we have: const carSpecs: [number, number] = [1907, 3354],

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

Eiffel

Subject

Year:

Month:

Date:

const carStats = {horsePower: 400, weight: 3354}; by looking at object, we can

immediately see what these fields are, but tuple is much harder to find out what

is going on.

5

2022/12/20

10

07 The All Important Interface

01

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 types, and describe

property names, and value types for objects. As we said, interfaces are never

custom types.

25

02

we create interfaces.ts type, then: const civic = {name: 'civic', year: 2009},

Eiffel

breaks true; then if we had a printVehicle method that takes `abc:circles.org:Vehicle`

`printVehicle(Vehicle vehicle)` { name: string; year: number; broken: boolean } or 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 { name: string; years: number; broken: boolean }` 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

Eiffel

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

5

functions, interface Vehicle { name: string; date: Date; broken: boolean; summary(): string }

any object that wants to be vehicle, must have all these fields.

10

05

There is two more pieces of syntax, do our vehicle need all these properties?

15

if we say: interface Vehicle { summary(): string } then: const oldCivic = { name:

'civic', year: new Date(), broken: true, summary() { return this.name } }, our

20

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

25

have summary? additional properties in oldCivic does not matter, because

Vehicle now only checks for summary method, we change its name to Reportable
Eiffel

06

now we create another object, const drink = { color: 'brown', carbonated: true,

sugar: 40, summary() { return 'My drink, has ' + this.sugar + ' sugars'; } }; now we have

5

an interesting thing: drink & oldCivic both implement summary() method, so they

are both of type: Reportable, we can use both with: printSummary(reportable: Re-

10

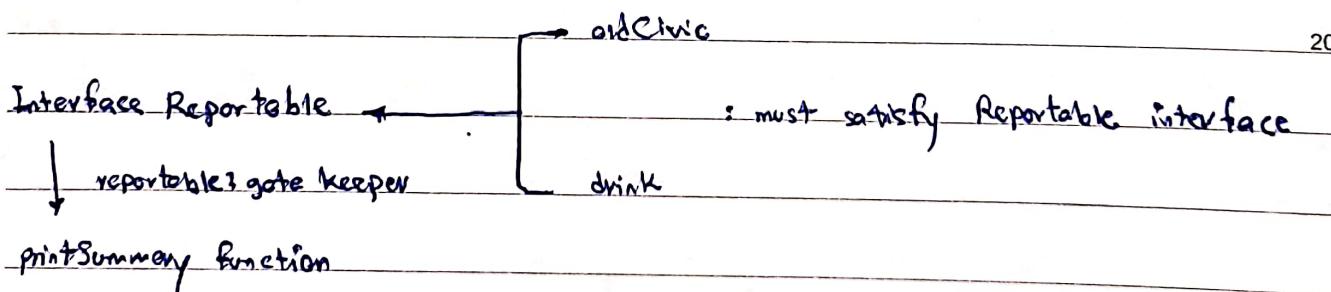
portable), So the point is with single interface, we can use different objects.

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

15

07

So our general plan with interfaces is:



20

General strategy: create functions that accept arguments as type interface; then

↓ reportable gate keeper

printSummary function

25

objects/classes decide to implement that given interface. Every application we

Eiffel

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

5

10

15

20

25

08 Building Functionality with Classes

01 classes

Class is blueprint: create different objects: fields (values) & functions, 5

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

class Vehicle { drive(): void { log('rrr') } } : blueprint, run this method: directly X10

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

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

02 basic inheritance

20

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

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

25

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

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

Eiffel

Vehicle: super class or base class,

03 instance method modifiers

5

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

restrict: access, default: public, if you don't add: modifier: public, public!

10

called: anywhere, anytime, priv: other methods! enact some class, make drives:

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

15

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

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

20

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

case:

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

25

other developers: can, protected: just like & private, different: access methods

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

Eiffel

in Car

in class Cov: super.drive => chugga chugga, this.drive => vroom, But: super.honk => beep, this.honk = beep,

04 fields in classes

adding 2 fields to the Vehicle => these are actual properties, none of

property, then annotation, color: string; => Err: not initialized, [mav: did not give err, until run], color: string = 'red', m/cov: 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, today: constructors, even empty: we should put it,

modifiers, effect: variables: some: functions,

Eiffel

05 fields in typescript

we know how fields work in class, how about inheritance? (or extend Vehicle)

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

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

: error! must contain: super(BM), maybe we put: super('red'), we don't want:

word code => 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: int

25

different classes, work together! help of interfaces,

Eiffel

08 Design Patterns with Typescript

09 Design Patterns with Typescript

01 app overview

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

ts

use ^ feature: really reusable code, Randomly Generate!

User

: then:

Company

10

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

: ts checker, inside browser, side npm instead of parcel-bundler

15

02 bundling with parcel

mk div maps, parcel-bundler → index.html → script of index.ts : parcel!

as soon as I sees, ts file, compile → JS, replace this script tag, maps → src div,

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

25

03 project structure

randomly generate

collection of different files, each file: class, class: one thing,

USER

→ share on

Company

map

our classes: likely: USER, Company, map, SRE div: indent.ts

company

user

map

indent.ts, User.ts is why uppercase: file: primary purpose: create & export

a class, indent.ts, not going to export any class, create class, user.ts: class or?

04 generating Random data

15

user properties: user name, location object: latitude, longitude, we are going

to: randomly generate these props, initialization: inside constructor, fakers:

20

generate fake data, fakers: take care of: all data: in our app, to access: module:

we have just installed: import faker from 'faker': could not find declarations

25

module: faker,

05 type definition files

JS Library

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:

in JS

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

anios: we didn't specify: Type Definition files, we have to install: type def: 10

manually: if is not: installed, once we find: we have to: install: type def file:

@types / library name? => @types/faker, Definitely Typed: naming, those files:

extremely: small, rest majority of time: you have to install: type definition file,

20

06 using type definition files

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

25

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

in user class: doing some: initialisation, in user: lat&long: object not initialized

Eiffel

automatically, `log(location)`: null or undefined, `this.location`.~~let~~ `zPL` => error

cannot read `let` of undefined, to initialize `let` & `mag` => initialize `location` before,

`parseFloat`: takes: string => return: number,

5

07 export statement in typescript

10

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

does anything with class, instead: export it, `export class User {}`, indents:

15

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

safely import several, `export default 'red': User.ts`, import red from 'User'

20

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

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

25

08 defining a company

repeat same process: create `Company.ts`, `companyName: string`, `catchPhrases: string`,
Eiffel

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

or 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=

② → status = 200,

to google maps integration

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

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

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

library

npm

type definition: how third party, works, type def: not only: for modules, but also:

script tags: added in: directly in 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,

5

11 exploring type definition files

10

Map instance, new google.maps.Map(): Err, wants two arguments: Element,

options: opt? → optional, type: second arg: MapOptions, different options to

15

customize map, ?: question mark: optional property, {zoom: 1?}, also:

center property: must satisfy: catch literal: ~~ctrl~~ ctrl + click on it: should have

20

{lat: number; lng: number},

12 hiding functionality

25

focus on: imports, company: Reference companyName catchPhrase lat/lng

user : Reference name lat/lng

Eiffel

googleMap: setZoom, setCenter, setStreetView, getHeading, ... methods we

enforce in our index.ts, other engineer: use these methods: break our app,

ideal things, enforce: map: addMarker, so googleMap: own its dangerous functions

hidden, only thing with custom map, create, add marker,

10

2022/12/21

Q3 why use private modifiers? here is why

to hide: create, custom map, capsule google map, enforce minimum amount of functionality,

create: CustomMap.ts, export custom map, hide, instance, google map, prevent,

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, completely eliminated:

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

14 adding markers

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

addUserMarker, addCompanyMarker, import User, Company types, creating time⁵

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

new google.maps.Marker({ map: ___, position: {lat: userLocation.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: hrs: lat&long,

16 one possible solution

25

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

properties

using or operator: to take a look: at two types; you can reference if both user

8 Company: here that prop, result of or operator: limit properties, there is still:

downside, what happen on future, if: GrLot \rightarrow display on map, or Park class \Rightarrow 5

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 solvable,

very tight coupling, btw our classes,

15

17 restricting access with interfaces

focus on best solution, prev sol: custom map: depend upon: User, Company, GrLot,

20

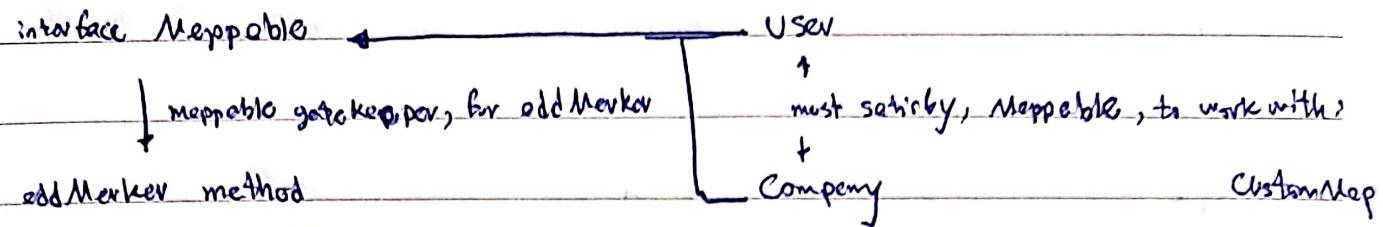
Park, rather than: custom Map: depending on classes! Hey User: you have to satisfy!

Map requirements, invert dependency, if you want to, have marker! to have

25

location & lot, number; long, number? property, \Rightarrow make use of: interfaces,

invert dependency: rather than: custom Map depending: other classes: User & Company:
Eiffel



a value I can have, multiple different types User and implements Mappable so

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 complain, behind

complains

the scenes: inspecting what type user is, because User & Company have been

20

{ut, long} → no error, no need extends (opposite of geno), 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,

Eiffel

InfoWindow: is the popup, add event listener, adding popup: a bit complicated

so: see the documentation, InfoWindow wants a content: be displayed,

now: static text: 'Hi there!', 5

20 updating interface definitions

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

words, we want: mappable, responsible for: this content, adding new requirement's

for mappable, adding: markerContent(): string \rightarrow Mappable, implementing: in:

(User & Company, descriptions: can be: html,

21 optional implements clauses

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

\Rightarrow Err: indent's: in User & Company, entire point of ts: help us find errors, part of

errors, index.ts is not best place to see err msg, seeing err in User & Company,

inside customMap: export Reportable, then: in User: class User implements -

mappable, by adding implements: saying ts: helps make sure: User implemented?

correctly, now: still: seeing err msg: inside index.ts also: in User, implements:

not required: etc all, reason: fail to: properly: implement: interface: get to true

sre of err,

15

22 app wrap up

20

reviewed important concepts: we learned, if: other engineer: open up: index.ts:

25

User: name, location, Company: name, location, catchPhrase, CustomMap: addMarker(),

we restricted: access to googleMap, duplicate Definition → first approach:

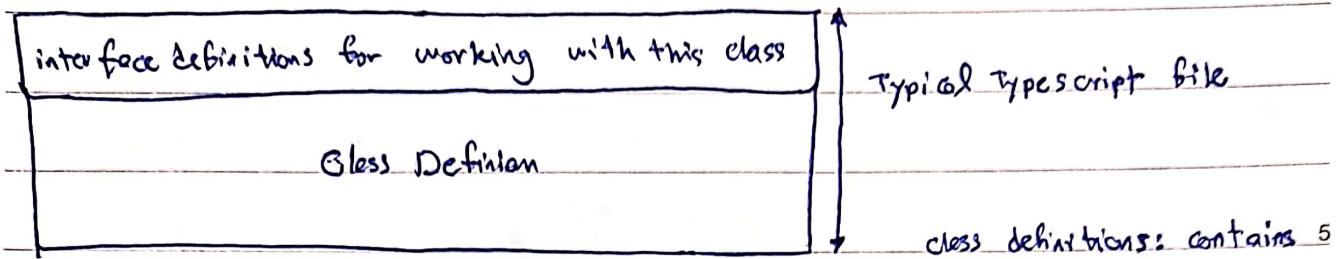
many different types, bad thing: set up dep: b/w customMap and all different

things, wanted to be on map → best sol: instead) customMap: implement, other classes:

Eiffel

implement,

big feature of ts: interplay b/w classes & interfaces, how we are gonna code,



methods, to work with class, if given method: has to receive: som obj: work

correctly, we are going to specify interface, very low coupling, help ts: 10

show up errors: optionally, use implements, ts: look, will ensure we implement

properly, that's it, three important things: 1. restrict api service area, 2. use interfaces

to manage dependencies, 3. find errors: correct notation: implements clause,

Subject

Year:

Month:

Date:

5

10

15

20

25

Eiffel

08 Building functionality with Classes

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,

Eiffel

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

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

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

base class

method has been overwritten. Vehicle in this example is ~~parent~~ or super class.

10

Q3

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, anywhere, 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

Subject

Year:

Month:

Date:

another error: `drive()` in `Vehicle` is public, but `driver()` in `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 5

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 can only be accessed in `Car`.

class Vehicle {

class Car extends Vehicle {

15

`drive(): void { log('chugga chugga')}`

`drive(): void {`

`protected honk(): void { log('beep')}`

`log('vroom')` }

}

}

20

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

'beep'.

25

04

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

Eiffel

then annotation \Rightarrow color: string; But we may get not initialized error,

we can ~~not~~ 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 5

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's

of constructor is empty; we should put it.

20

05

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

\Rightarrow constructor: provide color string; then error goes away. Car has no constructor,

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

Subject

Year:

Month:

Date:

public wheels, number) {} ; then we get error, it must contain super call, maybe we

put: super('red') ; but we don't want to hard code : constructor(public wheels:

number, color: string){ super(color)} ; Note we didn't put public before color, so

These explanations were very quick, but we will back to them in future.

10

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

15

interfaces.

20

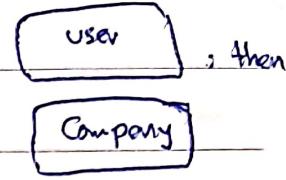
2022/7/21

09 Design Patterns with Typescript

25

we said we spent 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 nearly
Eiffel

reusable code. So our project is as follows: Randomly Generate! 

show them on the map. Every user/company, has a location. By doing this project

we get more familiar with classes. We use parcel-bundler, that is a ts executor

inside browser. ^{sudo} npm install -g parcel-bundler.

10

Q2, Q3

we make a maps directory. Then install parcel-bundler, and with parcel index.html

we can put ts file in html tag, it will compile ts file to js and replace

the tag. So we have, maps > 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

Eiffel

indent.ts, user.ts, Company.ts. if file primary purpose is create & export a class,

we write it with uppercase letter, but indent.ts is not going to export any
class.

04

Now we want to generate random data, User properties are user name, location

which is a object that contains lat & long, we are going to Randomly generate

these 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

Eiffel

see 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 angular, we didn't specify type definition files, we have to install type def file manually, if it is not installed. For installing type def file we use this: `npm install @types/library name` \Rightarrow `@types/faker` They are in Definitely Typed, these files are extremely small, vast majority of time, you have to install type definition file.

20

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 extension for type definitions files, if you see carefully, you will see latitude and longitude are of

Eiffel

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 lettings

we should initialize location before! location = {lat: parseFloat(faker.address.latitude)}

9.00.0

10

Q7

we want to see what does export statement do in TS. we completed our user, now

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 if 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.

5

2022/7/21 22

08

Repeating the same process, we create `Company.ts`, that has: `CompanyName: string`

`catchPhrase: string`; and `location` with `letting`, we initialize them with `faker`.

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

`parseFloat()` function.

20

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 is: key = ; to check everything is ok, we go to network section,

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

10

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

google; then it should show: {maps: fm?}, We usually install dependencies,¹⁰

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 google15

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 work²⁰

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

added in directly to html file. we instead: @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?: \Rightarrow 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 counterProperty must

satisfy letting literal; we don't 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: interface; Company reference

have these properties: companyName, catchPhrase, lat and lng. For user 2 name,

lat, lng.

Eiffel

setZoom, setCenter, setStreetView, getHeading, ... these are methods we expose in our imports, 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 encapsulate google map and expose 15

a minimum amount of functionality we create CustomMap.ts; then export customMap

so no hide instance 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 variable google; because it is wrapped

14

We want to write code for adding markers, first we write bad code, to implement

addMarker(); Then refactor into a better approach; addUserMarker(), addCompany-

5

Marker(), 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

10

adding a marker for user we have: new google.maps.Marker({map: ..., position: \$

let userLocation, let my: user.location.lng ??})

15

15

So we completed addCompanyMarker & addUserMarker; they look so similar, so

20

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 let&ing.

25

16

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

Eiffel

Company): So what does or() do? By using or operator, T3 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

15

17

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

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

Rather than customMap depending on classes we say: Hey user & company,

25

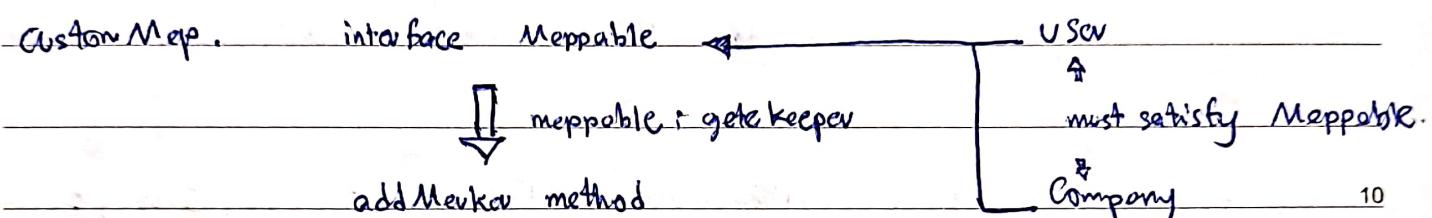
You have to satisfy Map requirements; and this is inventing dependency. If you

Eiffel

want to have marker, you should never use location: {lat, number, long, number}

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

Custom Map depending on other classes, other classes depend on interface of 5



18

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

Custom Map 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, long} 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.

Subject

Year:

Month:

Date:

19

We want to show a pop up window, we want, when clicked on marker, a pop up

comes. We implement pop up with InfoWindow, and then add event listener to it.

5

Adding a pop up is a little complicated so we see the documentation. InfoWindow

wants a content to be displayed. We put static text 'Hi there!' for content.

10

20

We want to customize content inside InfoWindow, we want it to be user class

15

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

20

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

21

25

Our app is now feature complete, if we add color property inside Mappable,

we get Err: in identifiers. Entire point of ts is to help us find errors, and not Eiffel

of errors. But `indents` 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 adding⁵

implements, we saying its, help us make sure `User` implemented `Reportable`

correctly. Now we still see Err msg inside `indents`, but Also see it in `User`'s

Note that: implements is not required at all, th

15

22

we want to review the concepts we learned in this chapter - if other engineers

open up: indents 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

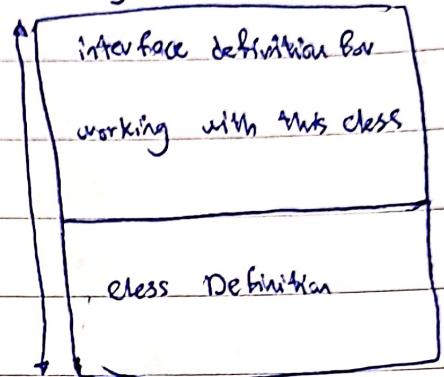
Eiffel

things that wanted to be on map. best solutions instead of customMap implementing

other classes; do vice versa. A big feature of TS is interplay between classes &

interfaces. This is how we are gonna code; Ts file

class definitions, contains methods to work with



class, if given method has to receive some object, to work correctly, we are

going to specify interface; that needs 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

10 More on Design Patterns

01 app overview

arr of numbers: [10, 5, 18, -3] → [-3, 5, 10, 18], string: 'projB' → 'aBjOP', linked list:

10 → -3 → 27 → 5 : -3 → 5 → 10 → 27, take collection: sort it, write out sorting algorithm

: one time → reuse

10

02 configuring ts compiler

15

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,

20

create build dir, src: typescript files, dest: compiled code, js code, configurations

compiler, tsconfig file: tsc --int → create tsconfig.json, every time, using tsc,

25

checks for: tsconfig: customized settings, to put ↗ ts → src ↗ js → build ↗ ! outDir, rootDir: relative path,

path, to where all codes can be find, outDir: where all compiled go, ↗ tsc ↗ i need
Eiffel ↗ run

to compile all code: From `rootDir`, `compile` into `outDir`, `tsc -w`; `watch`: all

files: inside `rootDir`, if anything changed: automatically compile again, or

Starting Compilation in watch mode, if changing smt: file change detected 5

23 concurrent compilation and execution

10

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 concurrency`

15

`nodemon`: run our project, concurrently: run multiple scripts, in script section: "start":

"build": "tsc -w", "start": "run": "nodemon build/index.js", "start": "concurrency"

20

`npm start:*` => tell concurrency: look at npm scripts, find every that starts with:

"start":, run them off, now: `npm start`: compile our code, rerun it! every time: a

25

change occurs,

04 a simple sorting algorithm

pretty solid: build setup => put them 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

indent's: 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 implementation

actual sort program, const length = this.collection.length; const length2 = this.length

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

however: it is number, we got: initial imp: put together,

25

07 two huge issues

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

Eiffel

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

[1, 3, 2]; strings: const color = 'red'; colors[0] // 'r' but colors[0] = 'y' // color, 'red',

strings: immutable, "Xaa" => sorted we expect: "aaaX", in JS: "X" > "a"; false, 5

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

as typescript is really smart

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

20

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

on both: common, working on: array, direct access elements, for string > direct access:

25

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

Q9 type guards

result of: `numbers | 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 is object, instanceof: every other value, created`

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

20

Q10 why is this bad?

`collection: numbers | string | LinkedList ...`, adding if statements in sort function,

25

and implementation: 15 sortable things, 18 if: doing almost same thing

11 extracting key logic

little bit of intermediate: refactor, two operations going to be customized,

based on: data structure, comparison swapping, correct comparison, helper function,

another class, Sorter.ts : sort(): { for() if (compare) { swap? } }, NumberCollection

: custom swap & compare: for numbers, Sorter, very generic, Sorter > abstract data

12 separating swapping and comparison

15

make: NumbersCollection.ts => data: numbers, implement compare & swap, we

should: make sure: there is length field; length(): number & return this.data.length

20

another way: natural: property not function: getter or accessor, get length()

: number { return this.collection.length? }, const c = new NumberCollection([1, 2, 3]),

25

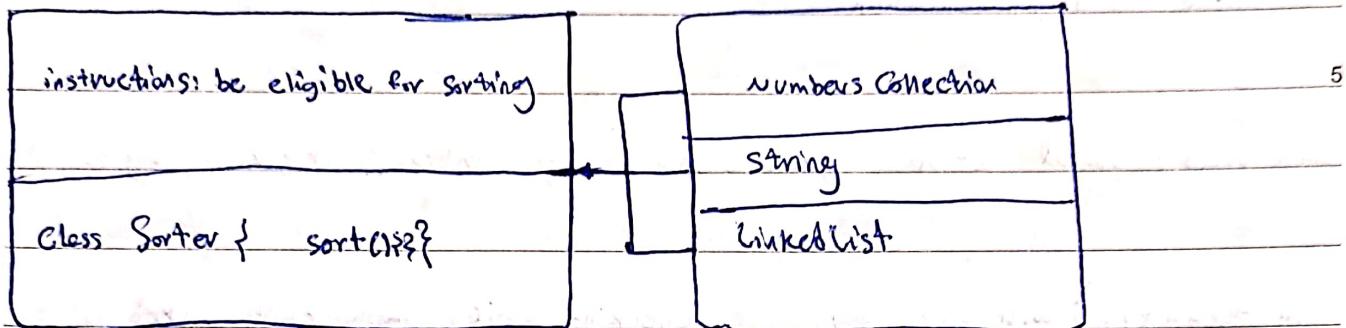
c.length, point of get, reference length as field, in Sorter: import

numbersCollection, use NumbersCollection methods: at sort,

Eiffel

13 the big reveal

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



10

instructions, length, swap, compare

14 interface definition

15

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

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

give them content, wege idea of meaning,

25

15 sorting arbitrary collections

creating: CharactersCollection.ts, comparisons using toLowerCase(), implementation Eiffel

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

16 linked list implementation

5

interfaces: useful, set up a type, also: set up: contracts: if you implement all

functionality, i will give you, not checking logic, implementing linked list logic,

10

for: comparison, length, swap

17 just one more fix

15

now: create NumbersCollection, create Sorter, call sorter.sort(), then print,

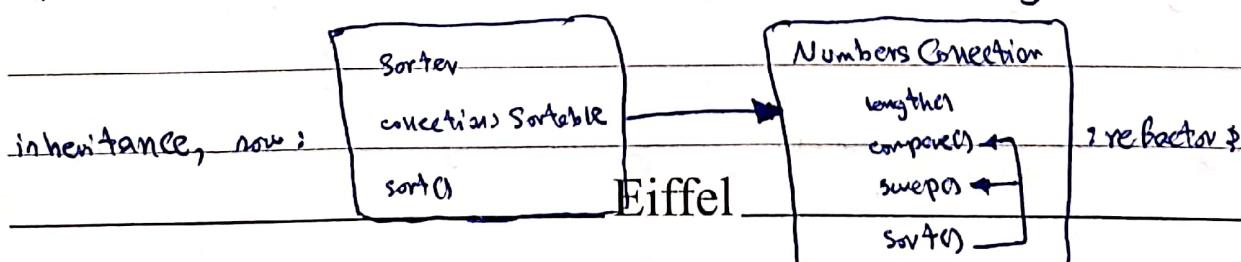
labourious, boring, better: numberCollection.sort(),

20

18 integrating the sort method

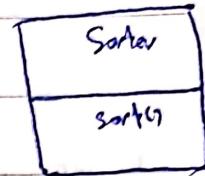
25

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



19 issues with inheritance

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



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

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

Sorter: directly, we just went, take sort method: copy it to child classes, 10

20 abstract classes

15

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

implement them, TypeScript Expectations

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

in reality: we implement methods: in NumberCollection

note: Sorter, we need to: make sure: TypeScript finds out: reality; Abstract

25

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

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

Eiffel

child classes) promise: imp & not imp methods,

21 why use abstract classes

5

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

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

10

we will treat: length: as property > field,

22 solving all issues with abstract classes

15

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

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

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

should call super, by using abstract class, you may think: we don't need 25

interface, interfaces, stwN super key in TS

023 interfaces vs abstract classes

we don't need interfaces anymore, quick compare & contrast: interface, set up

contract: b/w classes, different objs: want: work together, loose coupling;

abstract class/inheritance: set up contract: different classes, strongly

coupling: classes: together, Soter: without childs: childs without Soter, pointless,

15

20

25

Subject

Year:

Month:

Date:

5

10

15

20

25

Eiffel

10 More on Design Patterns

Q1

We want to implement a generic sorting algorithm. For example: array of numbers.

5

[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

10

want to write algorithm one time, and reuse it.

Q2

15

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

20

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 dst

25

to contain all compiled JS code. We configure ts compiler by changing tsconfig.json

ts:src

file, we create this file by: tsc --init. To put: Eiffel ↴ we change: js:build

outDir & srcDir: They are 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 watches

make and watch all files inside rootDir to see if any files changed, and it

will ~~not~~ compile it automatically again.

10

Q3

We have got automatic process for compiling our code, we can run our program by:

node build/index.js, 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 --

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": "node main build/index.js", "start": "concurrency":

npm: start: *". The last instruction tells concurrency to look at npm scripts,

then find every instruction that starts with: "start" and run them all, now: npm-

start: compiles our code and rerun it every time a change occurs.

10

04, 05, 06

we have a pretty 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.js, 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 coming from the collection.length, if you never

over it, its type is number.

5

07

Now we have some issues, Our sort() 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: numbers[0] = -1 => // numbers: [-1, 2, 3]

colors[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) charCode of 'X' is 88 and charCode 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[i] = this.collection[i+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, \Rightarrow 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'`: Linter signature in type `string` only permits reading.

25

09

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

Eiffel

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 some

for strings, but syntax 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 primitive¹⁰

types are: number, string, boolean, symbol. You can use typeof with for example

[] but, typeof [] : object. we use instanceof with every other primitive¹⁵

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

20

10

So why this code is bad? after a while, if we add more classes: collection1: number[]

string1: LinkedList ... and we should add if statement in our sort function,²⁵

and implementation for it. Suppose we have 15 sortable thing, then we have to

if statements, doing almost the same thing.

11

with a little bit of intermediate refactor, these two operations swap & compare²⁵

are going to be customized, based on the data structure. we want to extract

comparison, into helper function, on another class: Sorter.ts which has a sort()

method that is in the form: { for (1) { for (2) { 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 function²⁵

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 `this.length` as field, in Sorter we import 5

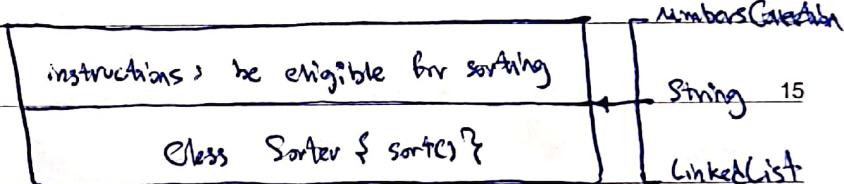
NumberCollection and use its methods at sort.

10

13

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

but we want to have:



instructions over to implement: length, swap, compare

20

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 5

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

Subject

Year:

Month:

Date:

Sense is giving content and we get idea of meaning.

5

10

15

20

25

Eiffel

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,
make std lib code re-use

02 project setup

in the past? package.json: scripts: "start: build": "tsc -w", "start: run": "node main

build/index.js", "start": "concurrently npm: start:*", build & run simultaneously,

mkdirs, npm init, 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 a read file, www.nodejs.org: File System, 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

node

for 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

10

at reading csv files

15

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

big string: parse: useful info, string $\xrightarrow{\text{split('n')}}$ lines: string[] $\xrightarrow{\text{map+split}}$ string[][]

20

inner array: one row data, map((row: string) \Rightarrow { return row.split(',') })
; string[]

each inner array: one forthell match \Rightarrow date, home team, away team, ... : home team

25

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: # times won game,

every match) man united home & won or man united away team & wins, for: 105

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

BB 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] ==

20

homewin =, that's better, we wrote: homewins every win! but: there is a draw,

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

25

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

07 using enums

replace all the identifiers: homeWin, awayWin, drew \Rightarrow const MatchResults

possible outcomes of match: { HomeWin: 'H',
awayWin: 'A', other engineers: looks like:
drew: 'D' }

why this object? shows smt important? instead, do we use enum

enum: hold numbers or strings, enum MatchResult { HomeWin = 'H', AwayWin =

'A', Drew = 'D' }, why use enum over object? signalling: other engineers:

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

08 when to use enums

20

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

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

drew}, in reality, when to \rightarrow compiles JS: not enum: object, primary goal: signal

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

Eiffel

closely related, known

at: compile time

25

only use enums when know all values at compile time, examples of when to

use / not use enums,

5

or extracting csv reading

10

we had some issues, src of data hardcoded, imagine we tried get football

data from api, if change src of data should change half of our current code,

15

create Csv.FileReader class: fields: filename, data: string[], Methods: read(): void,

read method: responsible for opening file, parsing info, now const reader = new

Csv.FileReader('football.csv'); reader.read(); if get info from api: just comment

20

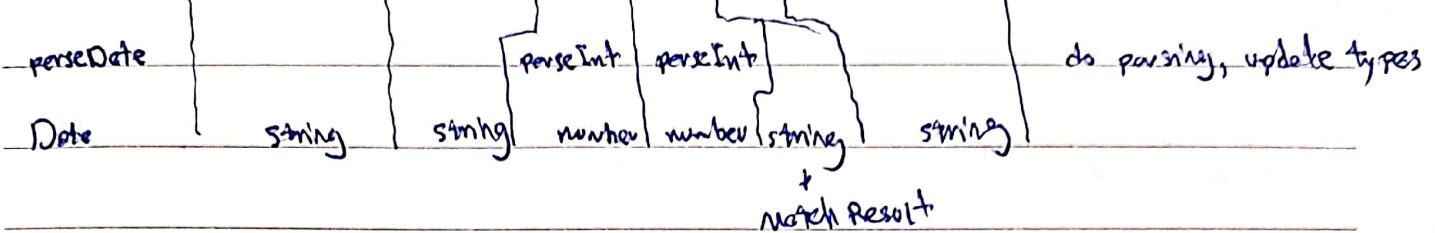
these two lines,

10 data types

25

Back to issues we had: our data are all strings although we have: date, number

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



5

11 converting date strings to date

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

10

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

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

15

0, So, pieces[2]-1, we use map: convert! then all: string → number,

12 convert raw values

20

add another map: when reading CSV: inside read method, inside map: for each

row (match), return [parseDate(row[0]), ~, parseInt(row[3]), ~], what about 25

Match Result?

Eiffel

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 is going to be one of the

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

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, after we have converted all correct type, because we return array:

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

before, [brown, true, 40], we want use tuple over array

```
{  
  color: brown,  
  boolean: true  
  sugar: 40}
```

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

Eiffel

25

15 not done with file reader yet

why? we created CsvReader.ts: 1. reusable src of info, 2b change to api:

delete a lot of code 2. can use on s future projects, But is it reusable

now? no, a lot of reference! match data, we want: refactor, make it:

reusable, cp ~~to~~ CsvFileReaders CsvFileReader.hk

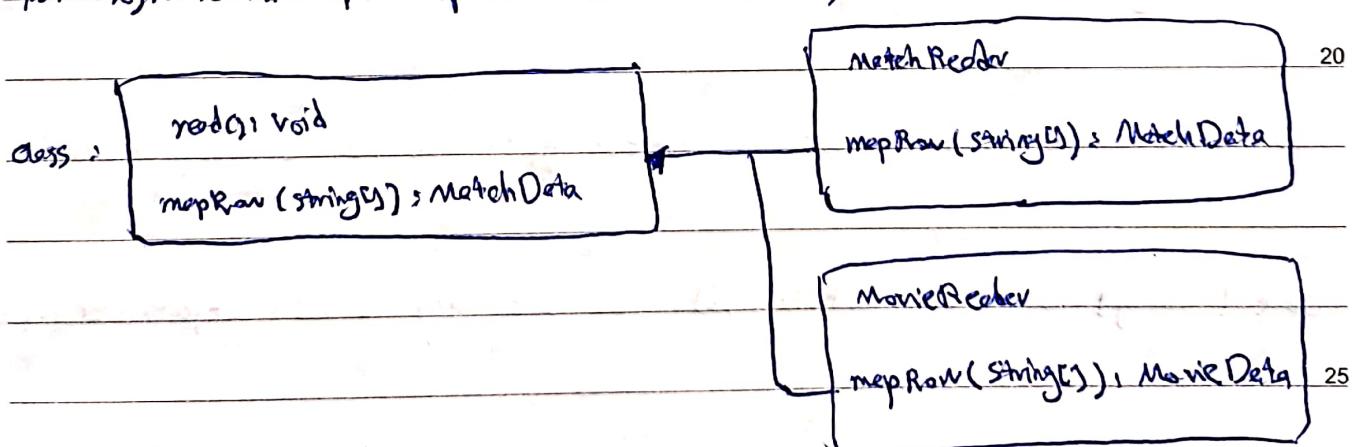
10

16 understanding refactor 1

15

first part, first map s in read method: generic, second map: custom logic,

put logic second map, helper function: mapRow, turn CsvReader: abstract



17 creating abstract classes

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

abstract mapRow (row: strings): 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: it's /varies

upon: football.csv, could change it to: any; not good idea; avoid any, emit 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,

20

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

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

25

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

imagine, when: add(2, 1) → 1 is replaced with b, make use: arg, customize
Eiffel

function behavior, with that in mind; class HoldNumber { data: number?;

class HoldString { data: string?; ... } just like functions, duplicated; for one

field, instead using generics: class HoldAnything<T> { data: T?; } now 5

const holdNumber = new HoldAnything<number>(2), same for string, reuse; es may

type as you want; any type, now we can replace: MatchData → generics: T, 10

why T? just convention, shorter of type, but we can name anything we want,

15

19 applying a type to a generic class

we can't use generics for anything; inside class, use for: add(a: T): T { ... }

20

so, we put: MatchData → MatchReaders: MatchReader extends CsvFileReader

This was: our Match Reader, refactor #1, finished, used > AbstractClass

25

20 alternate refactor

we achieved: code reuse, by: Abstract Class + Generics, pretty tricky; imp

Eiffel

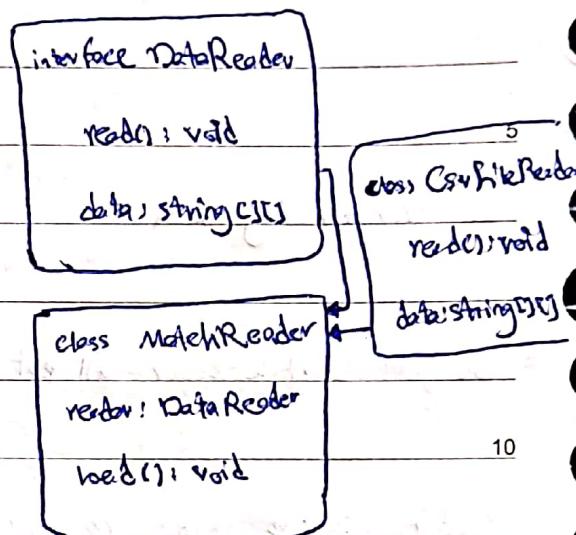
although its reusable, its hard to read, find out what's going on, more error

CsvFileReader & MatchReader, new dir, inheritance, mv: CsvFileReader.ts, back → 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,

Api Reader, Json Reader, etc,

21 interface based approach

nearly: opens interfaces: rather than: inheritance, create: MatchReader.ts: inside

it, create: class MatchReader, of: interface: Data Reader,

22 extracting match references again

start refactoring: CsvFileReader, CsvFileReader implements DataReader, removing Eiffel

all MatchData references from CsvFileReader $\xrightarrow{\text{move}}$ MatchReader, ensure match reader

data read, and converted,

5

23 transforming data

we got CsvFileReader's all put together, last thing to do: imp load methods Match-

10

Reader, inside read: reader.read(), then: data = data.map(λ) // convert to correct type
matches

now: our MatchReader & CsvFile Reader is ready,

15

24 updating reader references

last thing: update imports, to create instance of MatchReader, we should

20

pass: smt that + imp: Data Readers,

csvFileReader = new CsvFileReader(path)

matchReader = new MatchReader(csvFileReader)

two separate classes, where match reader

matchReader.load();

25

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

now we have all info: matchReader.matches,

new Match style (from Operator) نیز (مثلاً or if بدلانه) نیز است

football.csv file نیز Operator نیز (مثلاً if و if، if و if، if و if) هم این ورودی است

لیستی بقیه json API class'ı'ن src پذیرش می کند

10

2023/07/01

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, backed.

swept in different readers, inheritance: is on, composition: has a, in composition

we delegated loading op: outside obj,

25

26 more on inheritance vs composition

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

= put common in Rectangle, suppose: circular window, make super class Circle,

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

window: just differ: shape, abandon inheritance: use composition: create 10

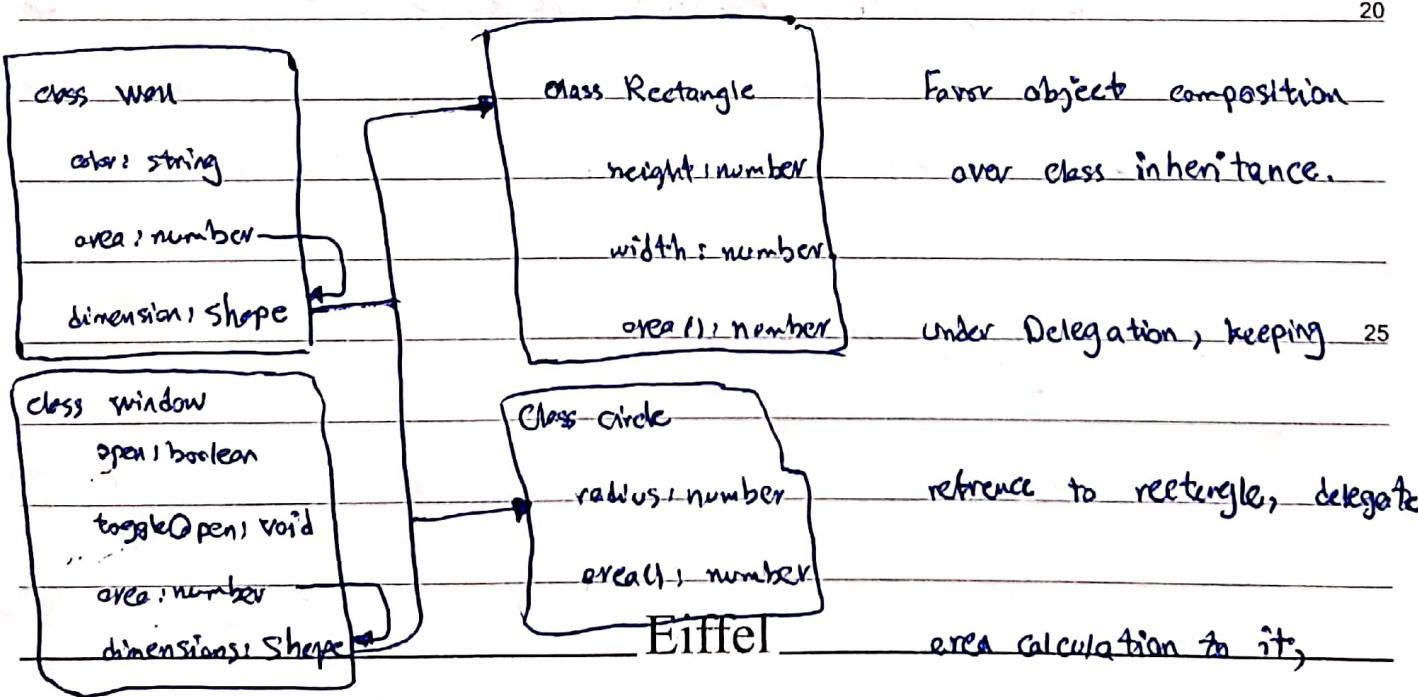
class Window & Wall: shape \Rightarrow dimension (property), interface Shape, Rectangle &

circle: Shape, easily swap, like our readers,

15

27 a huge misconception around composition

20



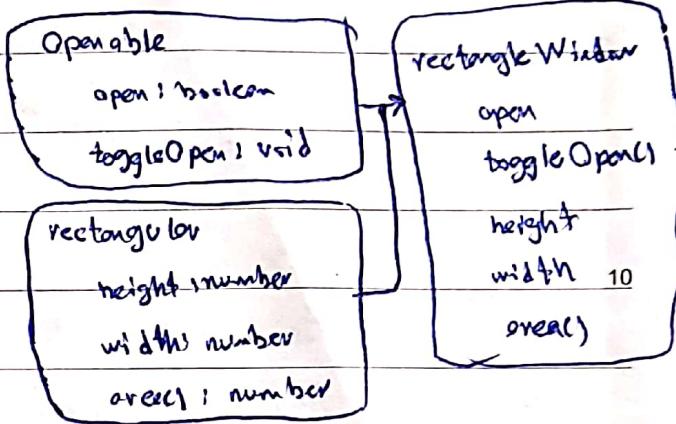
25

10

`Object.assign(state, config(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 teams: menu unit

interface Analyzer

interface OutputTarget

15

`run(matches: MatchData[]): string`

`print(report: string): void`

AverageGoalAnalyzer

winsAnalyzer

ConsoleReport

HtmlReport

analyser

20

outputTarget

`buildAndPrintReport` ⇒ delegate to: analyser & outputTarget

29 a composition based approach

25

all files: references MatchData; move it, its own file; other files ^{not} depend on!

MatchReader

Eiffel

Subject _____

Year: _____

Month: _____

Date: _____

match Data > needs MatchResult, so import it here, [01:25]

5

10

15

20

25

Eiffel