TypeScript Learning

27-01 TO 01-02

What is Typescript?

TypeScript is a superset developed and maintained by Microsoft. It is a strict syntactical superset of JavaScript and adds optional static typing to the language. TypeScript is designed for the development of large applications and transcompiles to JavaScript

Working:

1. index.html - The page which is displayed on the browser
2. app.ts - can have code in java script or type script which contains logic
3. tsconfig.json - contains all the compiler modules

Chapter 3

Basic types and Variables: Boolean, String, Number , Array, Enum
Additional Builtin types: Void, Null, Undefined, Never, Any

let key word :

allows user to let any variable hold any data type without specifying and has the added advantage of having to declare the variable with let before usage of the variable. This allows the user to be able to read the code efficiently

const keyword : similar to let but the value will be constant throughout the program

let variable_name : string = " hello"

let variable_name2 : number = 25

let variable_name3 = "hello" -> here the variable has taken up the data type of string due to the input and cannot be changed later on.

union -> let variable_name : string | number;

Here the variable can hold a string or a number , based on usage

Chapter 4

Adding Annotations to Functions

```
eg: function dullFunc (value1, value2){
                        return "hello";}
```

here we dont know what type is the function returning, or the type of the parameters passed to it.

```
-> function funFunc( score : number, message : string) : string {}
```

Here the score is a number , message is a string and the return type of the function is a string

Enabling //"noImplicitAny": true, will trace and throw an error while compiling if a parameter type isnt mentioned.

Default-intialised parameters:

```
function sendgreeeting (greeting: string ="good morning"): void{
        console.log(greeting);
}
```

if sendgreeting(); is called - will print 'good morning'
 if sendgreeting('good evening'); is called - will print 'good evening'

here default value of string is used to avoid null and implicity checks

Arrow Function (like lambda in python):

parameters => function; //syntax

eg: let sqaureit = x => x*x;
        let result = squareit(4);//16

let adder = (a,b) => a+b;
        let result = adder(2,3);//5

let greet = () => console.log('Hello');
        greet(); // Hello

if the function takes 0 or more than 1 parameters, the parenthesis () are required

```
function logMessage (message : string): void {

console.log(message);
}
logMessage('Good day');
```

|| simplified

```
const logMessage = (message: string) : void => console.log(message);
logMessage('Good day');
```

Chapter 5

class? interface? Examples

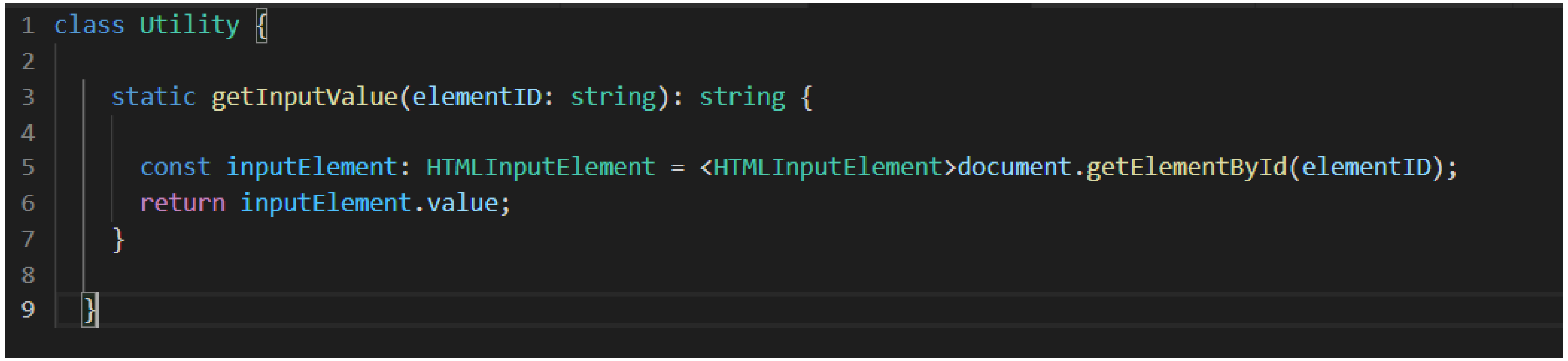
- Class: A class in terms of OOP is a blueprint for creating objects. A class encapsulates data for the object.
- interface:

how to reference interfaces and class files

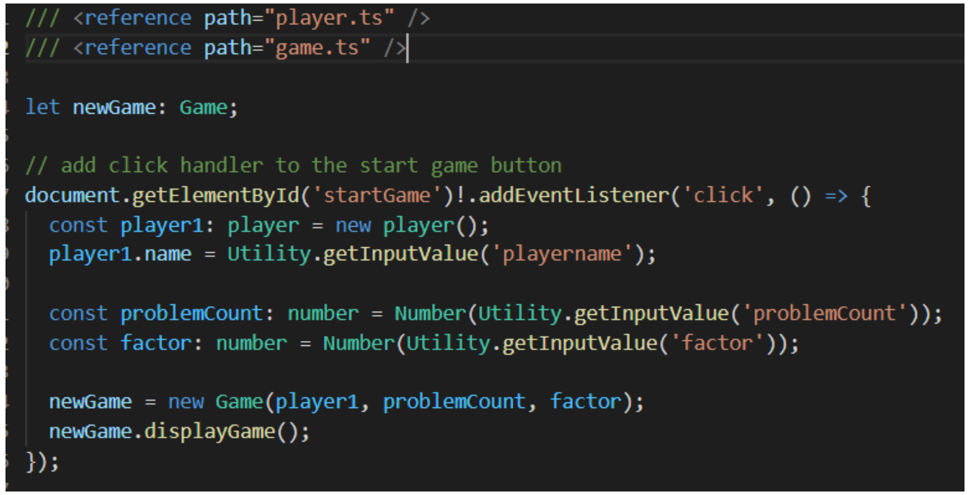
how to compile and send output to one js file instead of many

the changes made in all the files

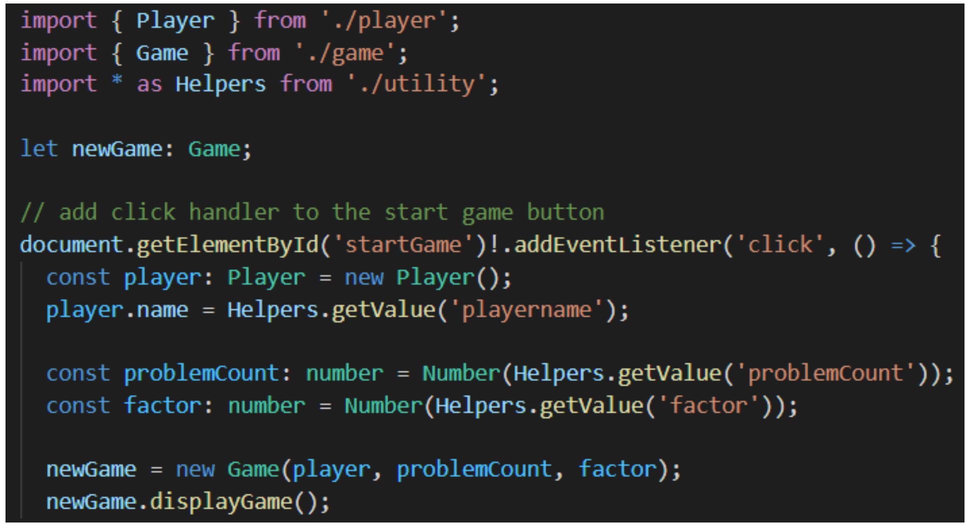
Class



```typescript
class Utility {

  static getInputValue(elementID: string): string {

    const inputElement: HTMLInputElement = <HTMLInputElement>document.getElementById(elementID);
    return inputElement.value;
  }

}
```

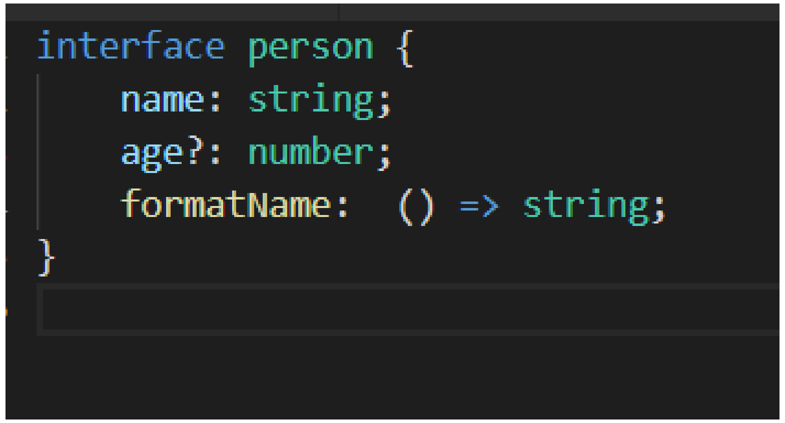


```typescript
/// <reference path="player.ts" />
/// <reference path="game.ts" />

let newGame: Game;

// add click handler to the start game button
document.getElementById('startGame')!.addEventListener('click', () => {
  const player1: player = new player();
  player1.name = Utility.getInputValue('playername');

  const problemCount: number = Number(Utility.getInputValue('problemCount'));
  const factor: number = Number(Utility.getInputValue('factor'));

  newGame = new Game(player1, problemCount, factor);
  newGame.displayGame();
});
```

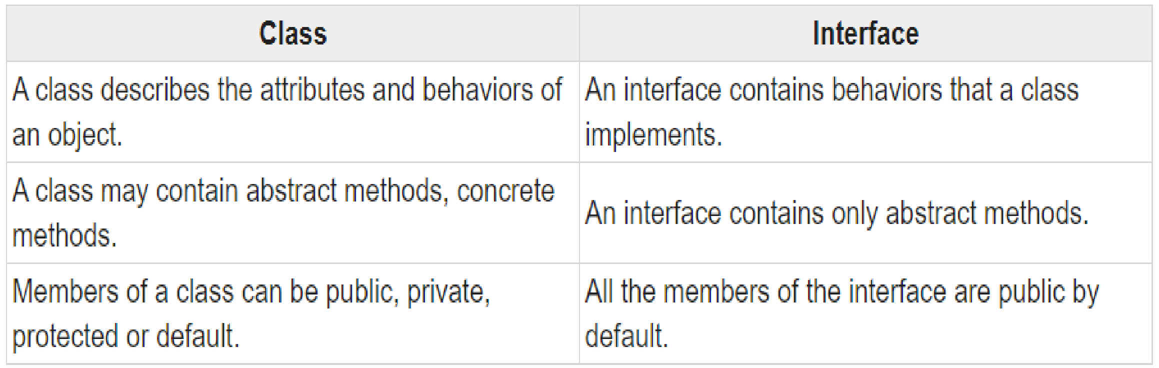


```typescript
import { Player } from './player';
import { Game } from './game';
import * as Helpers from './utility';

let newGame: Game;

// add click handler to the start game button
document.getElementById('startGame')!.addEventListener('click', () => {
  const player: Player = new Player();
  player.name = Helpers.getValue('playername');

  const problemCount: number = Number(Helpers.getValue('problemCount'));
  const factor: number = Number(Helpers.getValue('factor'));

  newGame = new Game(player, problemCount, factor);
  newGame.displayGame();
```

Interface



```
interface person {
    name: string;
    age?: number;
    formatName:  () => string;
}
```



Class	Interface
A class describes the attributes and behaviors of an object.	An interface contains behaviors that a class implements.
A class may contain abstract methods, concrete methods.	An interface contains only abstract methods.
Members of a class can be public, private, protected or default.	All the members of the interface are public by default.

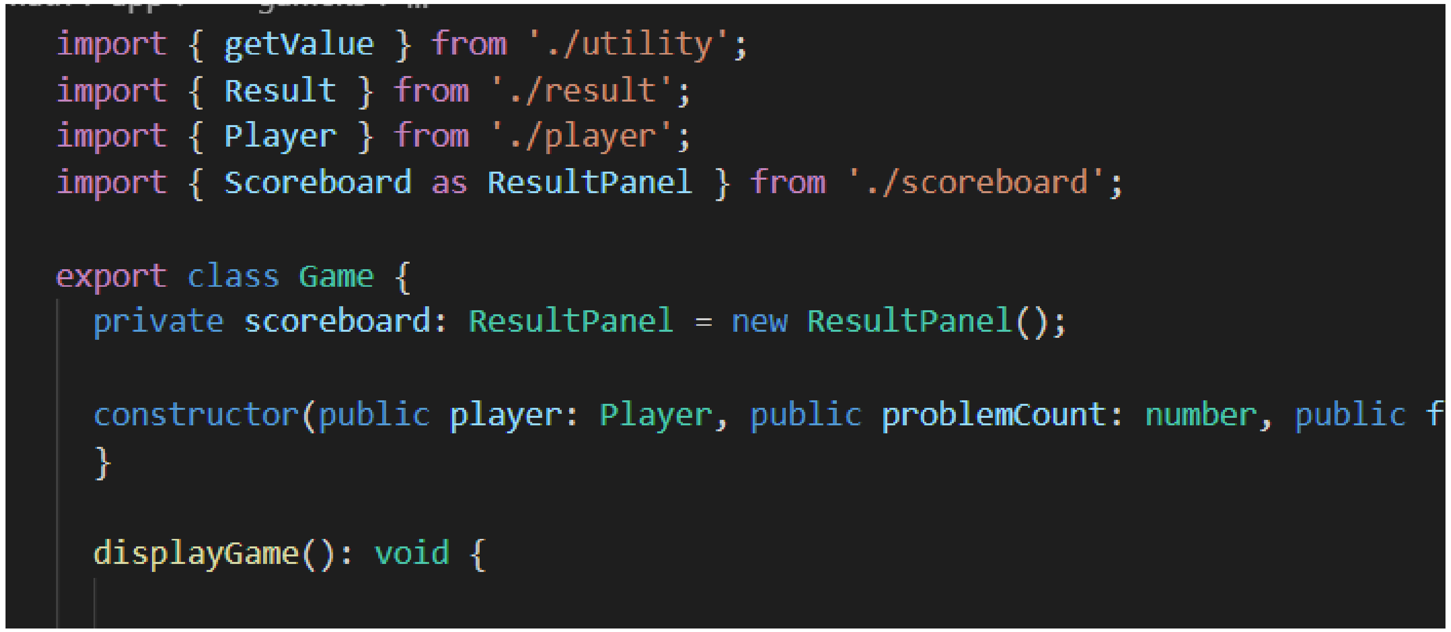
Reference to Classes and Interfaces

Demo and show

Also point out how , we make all the ts files compile into one file

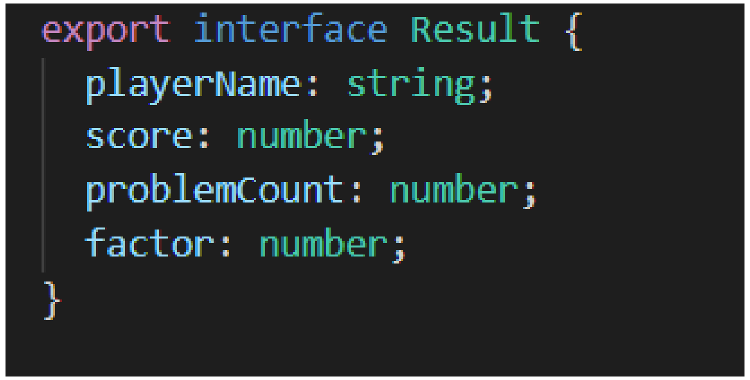
Chapter 6

Importing modules

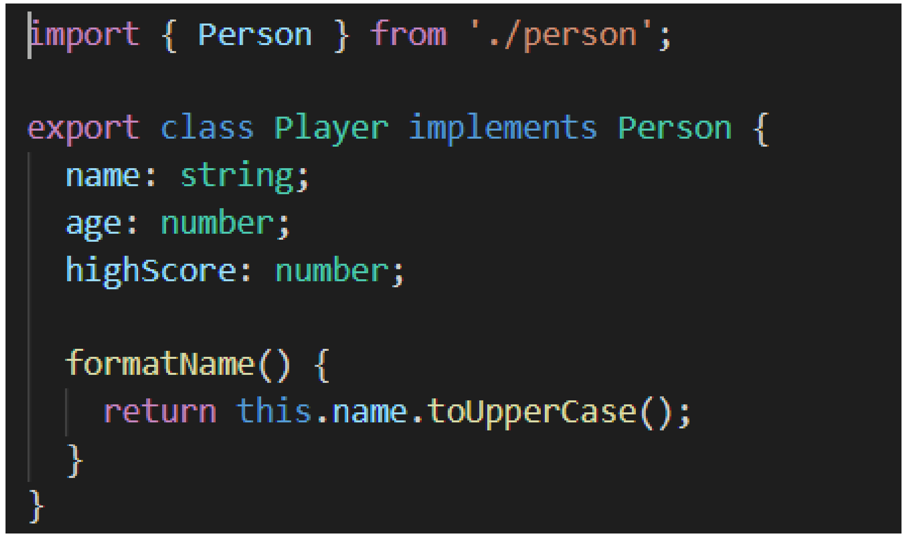


```
import { getValue } from './utility';
import { Result } from './result';
import { Player } from './player';
import { Scoreboard as ResultPanel } from './scoreboard';

export class Game {
  private scoreboard: ResultPanel = new ResultPanel();

  constructor(public player: Player, public problemCount: number, public f
  }

  displayGame(): void {
```

Exporting Modules

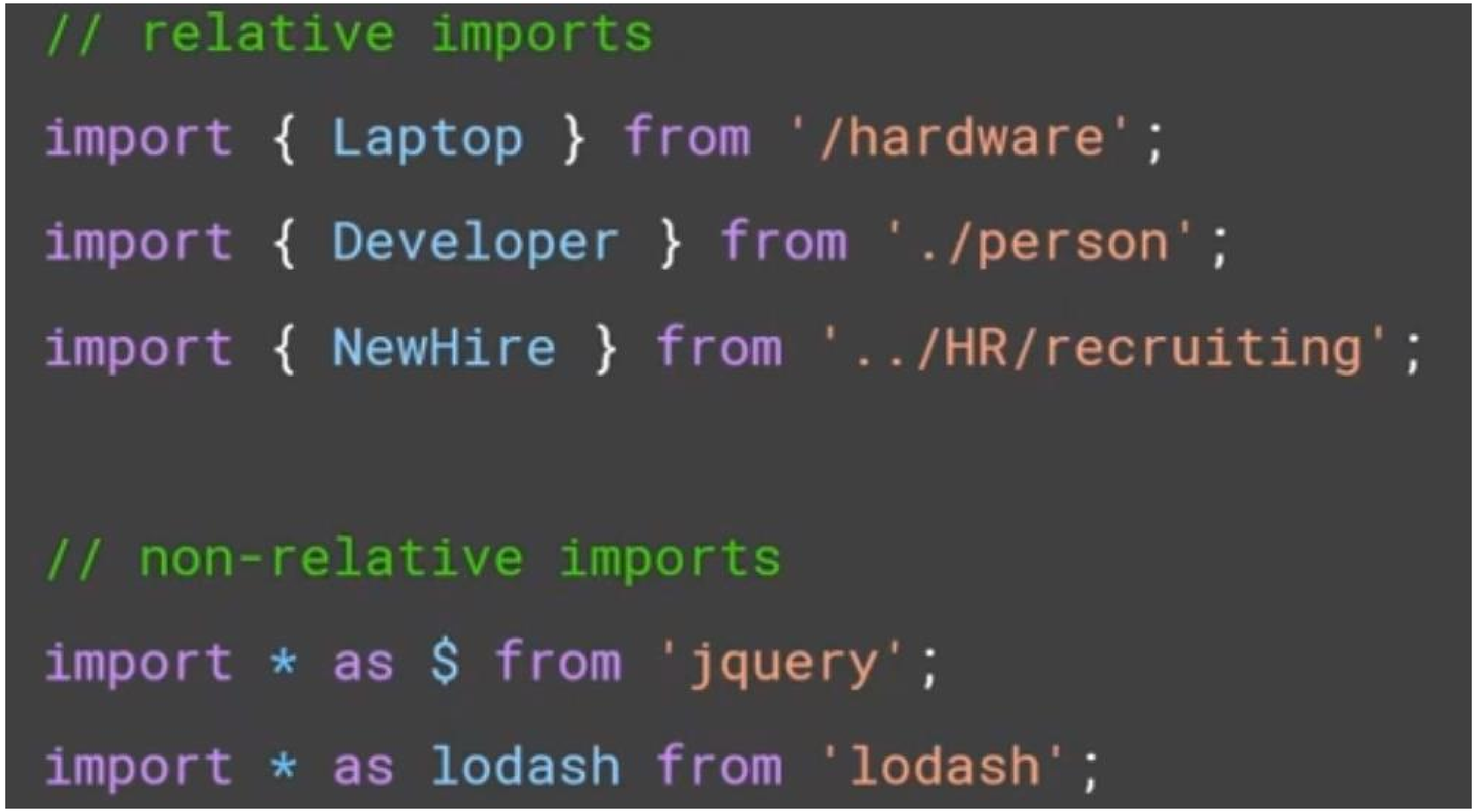


```
export interface Result {
  playerName: string;
  score: number;
  problemCount: number;
  factor: number;
}
```

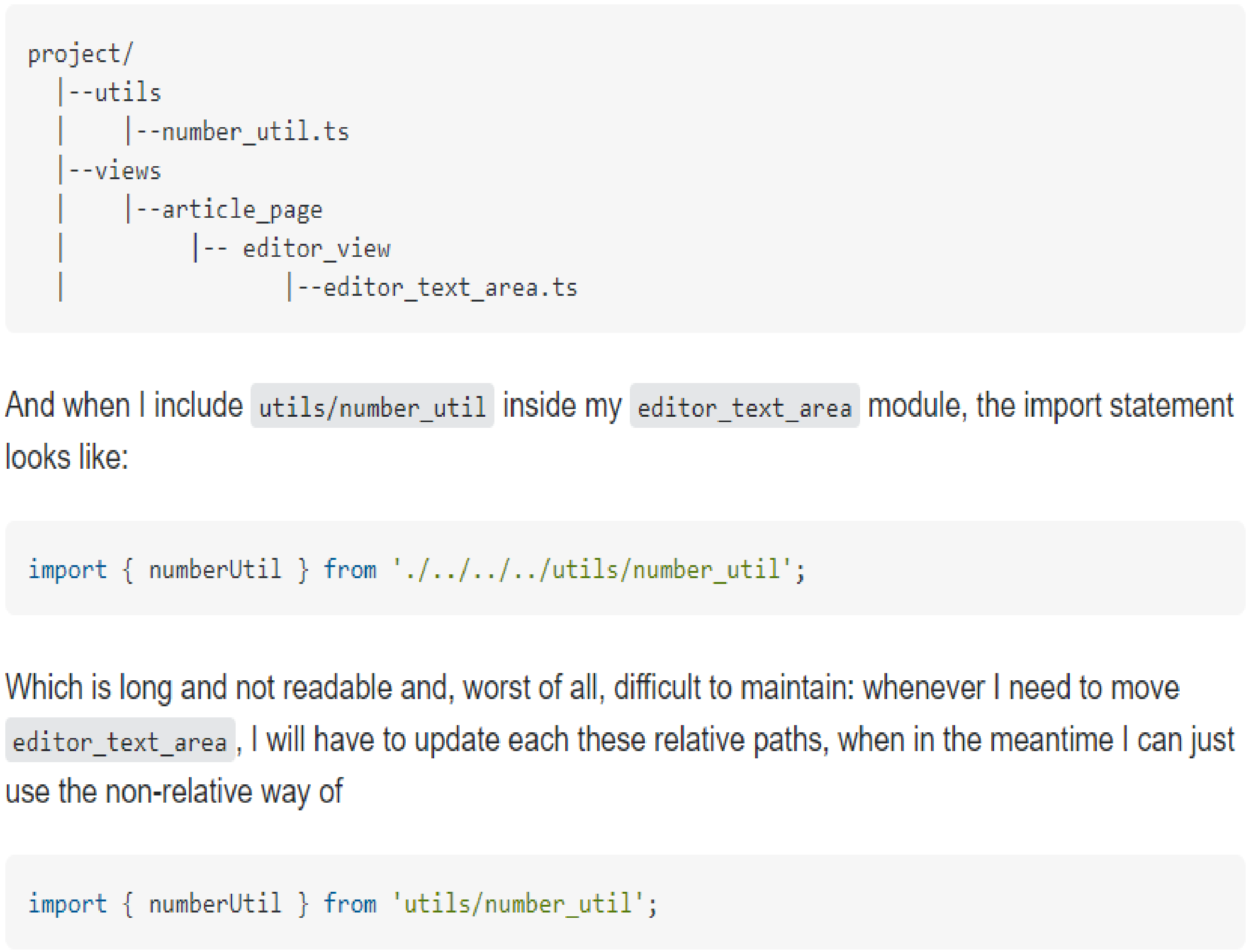


```
import { Person } from './person';

export class Player implements Person {
  name: string;
  age: number;
  highScore: number;

  formatName() {
    return this.name.toUpperCase();
  }
}
```

Relative and Non relative imports



```
// relative imports

import { Laptop } from '/hardware';

import { Developer } from './person';

import { NewHire } from '../HR/recruiting';


// non-relative imports

import * as $ from 'jquery';

import * as lodash from 'lodash';
```



```
project/
   |--utils
   |     |--number_util.ts
   |--views
   |     |--article_page
   |           |-- editor_view
   |                    |--editor_text_area.ts
```

And when I include `utils/number_util` inside my `editor_text_area` module, the import statement looks like:

```
import { numberUtil } from './../../../utils/number_util';
```

Which is long and not readable and, worst of all, difficult to maintain: whenever I need to move `editor_text_area`, I will have to update each these relative paths, when in the meantime I can just use the non-relative way of

```
import { numberUtil } from 'utils/number_util';
```