**Typescript:**

1. **npm install -g typescript**
2. **D:\>tsc --version**

**> Version 3.7.3**

D:\>cd an\*

D:\angular>mkdir ts-hello

D:\angular>cd ts-hello

D:\angular\ts-hello>code main.ts

D:\angular\ts-hello>

D:\angular\ts-hello>**tsc main.ts**

D:\angular\ts-hello>dir

Volume in drive D is Temporary Data

Volume Serial Number is E231-C2A8

Directory of D:\angular\ts-hello

09.12.2019 11:44 <DIR> .

09.12.2019 11:44 <DIR> ..

09.12.2019 11:44 100 main.js

09.12.2019 11:44 97 main.ts

2 File(s) 197 bytes

2 Dir(s) 83.385.982.976 bytes free

D:\angular\ts-hello>**node main.js**

Hello World

D:\angular\ts-hello>node main.ts

Hello World

D:\angular\ts-hello>code main.js

1. Using let instead of var

function doSomething() {

for (let i = 0; i < 5; i++) {

console.log(i);

}

console.log("Finally: " + i);

}

doSomething();

D:\angular\ts-hello>

1. **Declaring Variables:**

**let a: number; //** Data types with annotation

**let b: boolean;**

**let c: string;**

**let d: number[] = [1,2,3];**

**let f: any [] = [1, true, 'a', false];**

**enum Color {Red=0, Green=1, Blue=2}**

**//** Enum: Group of related constants

**let backgroundColor = Color.Red;**

**console.log(backgroundColor);**

**const Pet1 = "Mouse";**

**const Pet2 = "Cat";**

**const Pet3 = "Dog";**

**enum Animals {PetA, PetB, PetC };**

**let myPet = Animals.PetC;**

**console.log(myPet);**

**enum Color {kirmizi=0, yesil=1, mavi=2, mor=3};**

**let myColor = Color.mor;**

**console.log(myColor);**

1. **Pointing the variable type explicitly**

**let message;**

**message = "abc";**

**let endsWithC = (<string>message).endsWith("c"); *// adding the type of the variable explicitly as string***

***// without that its type was any and we could not see the properties such as endswith***

**console.log(endsWithC);**

C:\Users\musta\Desktop\LearnCoding\angular\ts-hello>tsc --target ES6 message.ts

C:\Users\musta\Desktop\LearnCoding\angular\ts-hello>node message.js

true

false

C:\Users\musta\Desktop\LearnCoding\angular\ts-hello>

**Note:** declarıng the var type does not change the var type it is just the way to inform the compiler.

1. **Arrow Functions:**

let log = function(message) {

console.log(message);

};

let doLog = message => console.log(message);

let doLog1 = message => console.log(message);

let doLog2 = () => console.log();

1. **Interfaces:**

**interface Point {**

**x: number,**

**y: number**

**}**

**let drawPoint = (point: Point) => {**

***// ...***

**}**

**drawPoint({**

**x: 1,**

**y: 2**

**})**

1. **Cohesion:**

In OOP: things that are related should be part of one unit. So that routes us to the classes.

1. **Classes:**

**interface Point {**

**x: number,**

**y: number**

**}**

**let drawPoint = (point: Point) => {**

***// ...***

**}**

**let getDistance = (pointA: Point, pointB: Point) =>{**

***//...***

**}**

**drawPoint({**

**x: 1,**

**y: 2**

**})**

**Class:** Groups variables (properties) and functions (methods) that are highly related.

interface Point {

x: number,

y: number

draw: () => void

}

**.**

**.**

**.**

**.**

**class Point {**

**x: number;**

**y: number;**

**draw () {**

***//..***

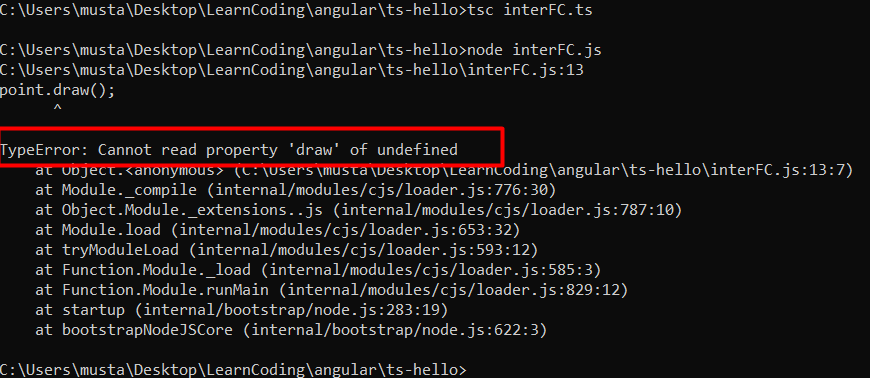
**}**

**getDistance = (anotherPoint: Point) {**

***//...***

**}**

**}**

****

class Point {

x: number;

y: number;

draw() {

console.log("X: " + *this*.x + "Y: " + *this*.y);

}

getDistance(anotherPoint: Point) {

*//...*

}

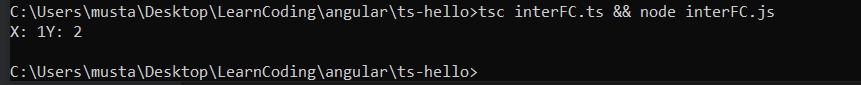
}

let point = new Point();

point.x = 1;

point.y = 2;

point.draw();



1. **Constructors:**

**class Point {**

**x: number;**

**y: number;**

**constructor (x: number, y: number) {**

***this*.x = x; *// initialize***

***this*.y = y; *// initialize***

**}**

**draw() {**

**console.log("X: " + *this*.x + "Y: " + *this*.y);**

**}**

**}**

**let point = new Point(1,2);**

**point.draw();**

**If you do not want to use the parameters in some cases, add ? to the right of variable in constructors**

**class Point {**

**x: number;**

**y: number;**

**constructor (x?: number, y?: number) {**

***this*.x = x; *// initialize***

***this*.y = y; *// initialize***

**}**

**draw() {**

**console.log("X: " + *this*.x + "Y: " + *this*.y);**

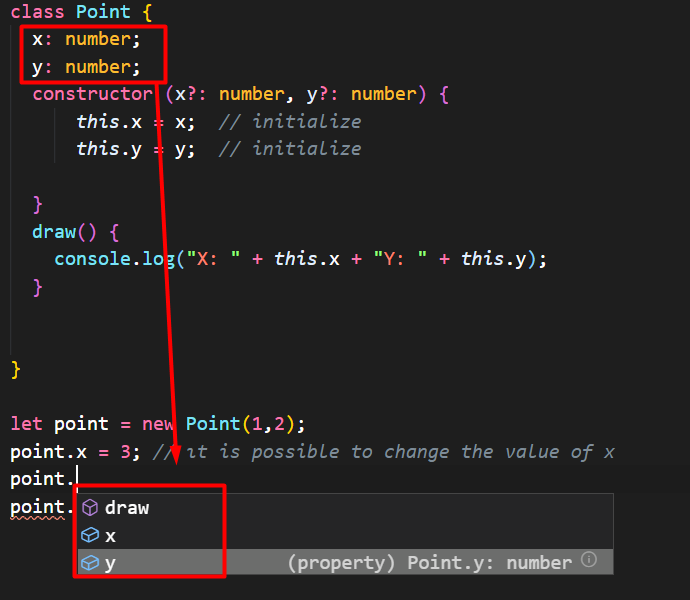
**}**

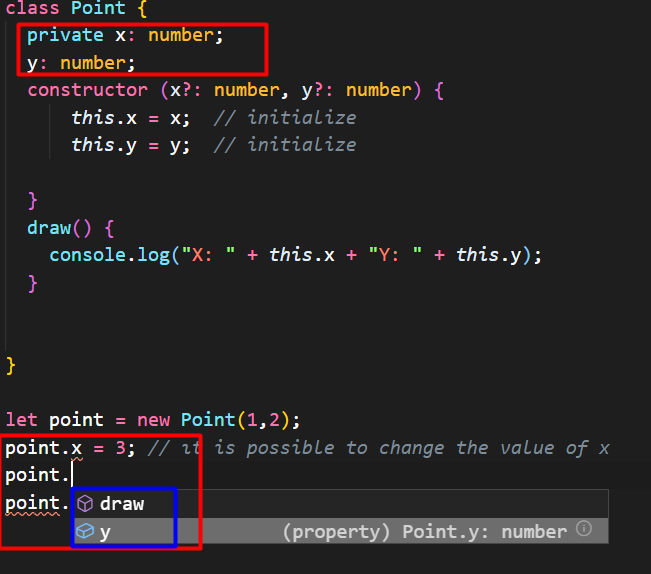
**}**

**let point = new Point();**

**point.draw();**

* **TypeScript Access Modifiers:**
  + **Public**
  + **Private**
  + **Protected**

****

****

**Private modifier avoids the coordinates to be changed.**

**If no access modifiers are set it is public by default**

**class Point {**

**x: number; // public**

**y: number; // public**

**constructor (x?: number, y?: number) {**

***this*.x = x; *// initialize***

***this*.y = y; *// initialize***

**}**

**draw() {**

**console.log("X: " + *this*.x + "Y: " + *this*.y);**

**}**

**}**

**let point = new Point(1,2);**

**point.draw();**

**12. Access Modifiers in constructors:**

After adding access modifiers in the constructor than we do not need to initialize declare the variables seperately and we do not need to initialize them in the constructors. , if you add modifiers than it typescript complier will generate a field with the exact same name and initialize it automatically

**class Point {**

**constructor (public x?: number, private y?: number) {**

***// this.x = x; // no need to initialize, if you add modifiers than it typescript***

***// complier will generate a field with the exact same name and // initialize it automatically***

***// this.y = y; // same as x above***

**}**

**draw() {**

**console.log("X: " + *this*.x + "Y: " + *this*.y);**

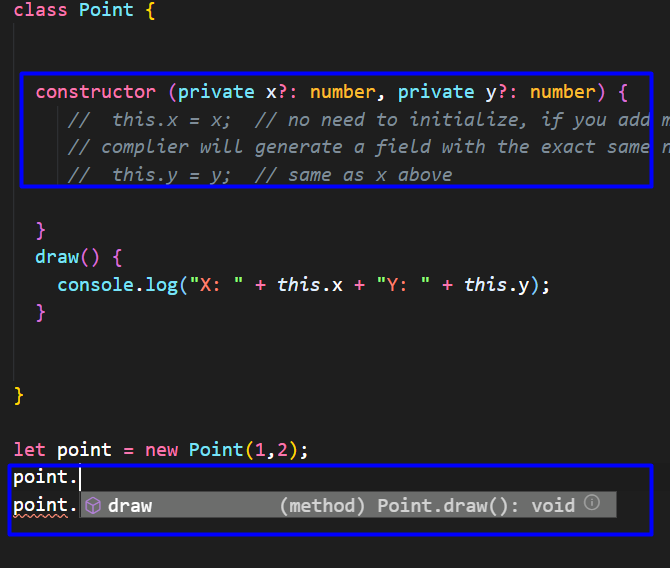
**}**

**}**

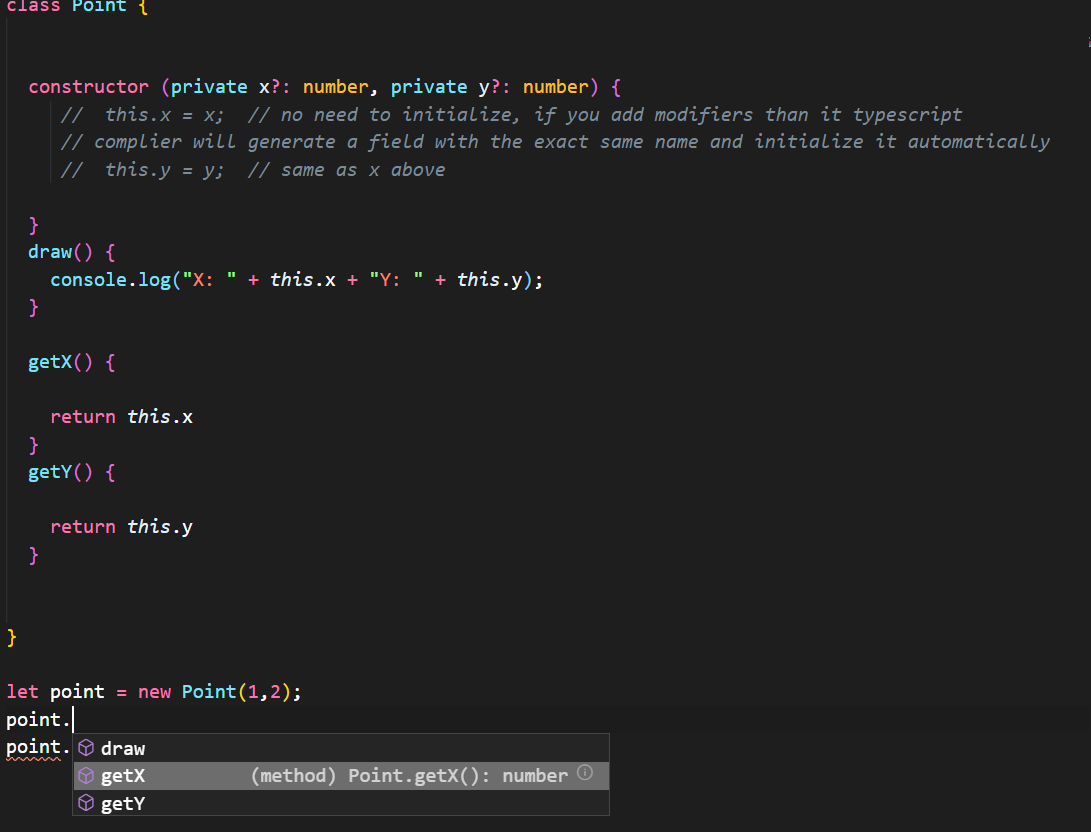
**let point = new Point(1,2);**

**point.draw();**

**13. Getters and Setters:** But we have a problem in the above şmplementation. What if we want to read the value of x or y coordinate? Becuse we set them as private!

****

**A simple solution is adding a get method.**

****

**As a further step, let’s assume we wanna allow the user to change the coordinates only within a certain range.**

**class Point {**

**constructor (private x?: number, private y?: number) {**

**}**

**draw() {**

**console.log("X: " + *this*.x + "Y: " + *this*.y);**

**}**

**getX() {**

**return *this*.x**

**}**

**getY() {**

**return *this*.y**

**}**

**setX(value) {**

**if(value<0)**

**throw new Error('Value cannot be less than Zero!')**

***this*.x = value;**

**}**

**}**

**let point = new Point(1,2);**

**point.setX(5);**

**point.draw();**

**Let’s define get and set methods as properties.**

**getX() {**

**return *this*.x**

**}**

**.**

**.**

**.**

**get X() {**

**return *this*.x**

**}**

**setX(value) {**

**if(value<0)**

**throw new Error('Value cannot be less than Zero!')**

***this*.x = value;**

**}**

**.**

**.**

**.**

**set X(value) {**

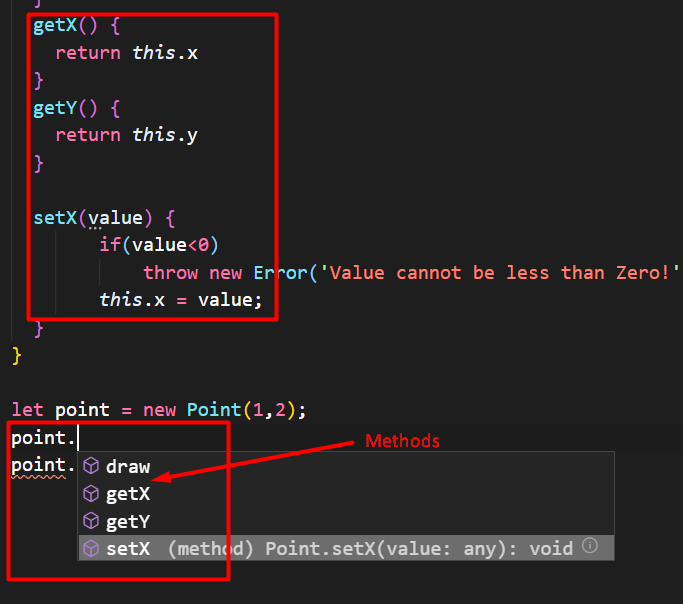
**if(value<0)**

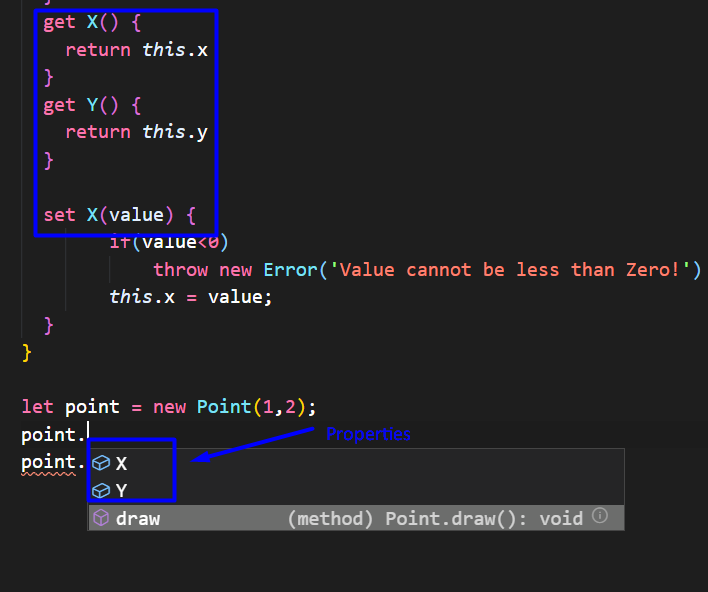
**throw new Error('Value cannot be less than Zero!')**

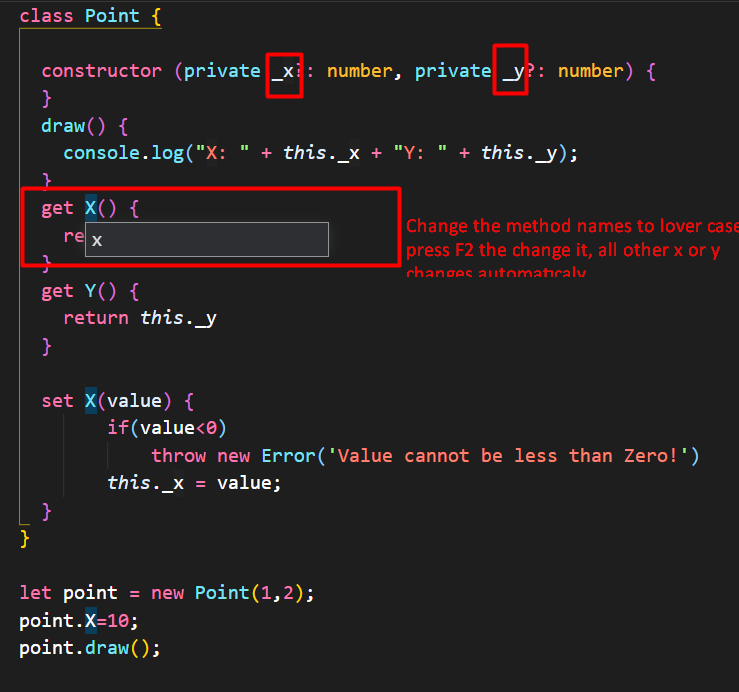
***this*.x = value;**

**}**

So what changed now?







class Point {

constructor (private \_x?: number, private \_y?: number) {

}

draw() {

console.log("X: " + *this*.\_x + "Y: " + *this*.\_y);

}

get x() {

return *this*.\_x

}

get y() {

return *this*.\_y

}

set x(value) {

if(value<0)

throw new Error('Value cannot be less than Zero!')

*this*.\_x = value;

}

}

let point = new Point(1,2);

point.x=10;

point.draw();

Lesson: A property looks like a field from the outside but internally it is exactly a method in the class.

**13. Export and Import:**

export class Toint {

constructor (private \_x?: number, private \_y?: number) {

}

draw() {

console.log("X: " + *this*.\_x + "Y: " + *this*.\_y);

}

}

import {Toint} from './toint'

let toint = new Toint(1, 2);

toint.draw();

