**Type Script Online Sessions by Sir Zia**

**Class # 1 : Structural Programming 11-Apr-23**

[**https://www.youtube.com/watch?v=ht8GZibRIA8**](https://www.youtube.com/watch?v=ht8GZibRIA8)

Type Script uses Structural Typing not Nominal Typing. Consider the following example :

// These two interfaces are completely transferrable in a structural type system:

interface Ball {

    diameter: number;

}

interface Sphere {

    diameter: number;

}

  let ball: Ball = { diameter: 10 };

  let sphere: Sphere = { diameter: 20 };

  sphere = ball;

  ball = sphere;

In Nominal Type this would have been an error. As Ball and Sphere are different interfaces in Nominal Typing. But Type Script uses Structural Typing. In Structural Typing although Ball and Sphere are two different types of Interfaces but Type Script compares its keys and their types. So if both have same key name and type, then object of both these types can be assigned to each other.

**Note :** We can also use type instead of Interface . Both are used to create type , there is a minor difference between them. For Example :

type Ball {

    diameter: number;

}

type Sphere {

    diameter: number;

}

We can also define an Inline Type as :

 // We can also define Inline Types as

 let ball: {diameter:number} = { diameter: 10 };

 let sphere: {diameter:number} = { diameter: 20 };

Consider we create another type that contains all keys present in Ball and Sphere and also contain an additional property

  // If we add in a type which structurally contains all of the members of Ball and Sphere, then it also can be set to be a ball or sphere.

  interface Tube {

    diameter: number;

    length: number;

  }

  let tube: Tube = { diameter: 12, length: 3 };

  //tube = ball;//Error

  ball = tube;

In the above created type Tube there are two properties one is diameter and 2nd one is length. From above mentioned example we can see that we can assign a tube type variable to ball but we cannot assign ball type variable in the tube type variable . Now as ball has only one key and does not contain the 2nd key so we cannot assign a ball type object in the tube type object. In opposite case we can assign tube type object to a ball type object as ball requires only one key diameter which is present in the tube type object.

Let us consider an other example of Structural Typing: Consider the example given below :

// Case 2a

let myType = { name: "Zia", id: 1 };

//Case 1

myType = { id: 2,  name: "Tom" };//Case 1: can only assign a type which has the the same properties. Object literals can only have properties that exist in contextual type

In above lines of code we have created a new type using Inline Typing. In the above example 1st we assigned value to myType while creating. Then we again assigned a new value to myType by providing value of both it and name. This is permissible as while assigning name and datatype of keys should match

Let us discuss our second case :

//Case 2a

myType = { id: 2,  name\_person: "Tom" };//Case 2a: Error, renamed or missing property

//Case 2b

//A type can include an index signature to explicitly indicate that excess properties are permitted in with fresh objects:

var x: { id: number, [x: string]: any };//Note now 'x' can have any name, just that the property should be of type string

x = { id: 1, fullname: "Zia" };  // Ok, `fullname` matched by index signature

In the case 2a the assignment failed as the name of the key is different .

In the case 2b we have deifned a variable 23:54

Here we should know the concept of Steal and Fresh.