**Type Script Online Sessions by Sir Zia**

**Session # 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, [y: 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 defined a variable type that includes an ***index signature*** to explicitly indicate that excess properties are permitted in with fresh objects. ***Note now 'y' can have any name, just that the property should be of type string***

We can use this syntax in case when we do not know the name of the keys in such cases we can define type as above. TS will accept the key name as long as the key signature is matched. I.e: it can have any key name but its type must be string. Here any is not type it is indicating that any key name can be assigned.

So if we now x = { id: 1, fullname: "Zia" }; it will not generate error as key name can be any name but its type must be string. We can use any key name instead of fullname for exmaple name , first\_name etc .

Let us discuss another case given below:

//Case 3

myType = { id: 2,  name: "Tom", age: 22 };//Case 3: Error, excess property

Here we are assigning myType variable three properties as age property is not available in it so this code will generate an error.

Rule for Fresh and Stale Objects

In TypeScript, the "freshness" of an object states to whether the object has been modified since it was last retrieved. An object that has not been modified is considered "fresh," while an object that has been modified is considered "stale."

Case # 1 : Case when same data type and key name object literal are assigned to a variable : Ok. No Error

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

let myType2 = { id: 2,  name: "Tom" };

// Case 1: Can only assign a type which has the same property types and names

// Rule is same for Fresh and Stale object

myType = myType2;

Case # 2a : Case when different data type or key name containing object literal are assigned to a variable Error as Renamed or Missing property. Rule is same for stale and fresh object

let myType3 = { id: 2,  name\_person: "Tom" };

// Error in case of Renamed or missing property

// Rule is same for stale and fresh object

myType = myType3; // This will generate error as key names are different

Case # 2b : Case when a type include an index signature to explicitly indicate that excess properties are permitted in with fresh objects:

var x: { id: number, [y: string]: any };

//Note now 'y' can have any name, just that the property datatype should be of type string

var z = { id: 1, fullname: "Zia" };

x = z;// Ok, `fullname` matched by index signature

Case 3: When an Excess property is assigned to a STALE object

var myType4 = { id: 2,  name: "Tom", age: 22 };

myType = myType4;

// No Error. As my type only required id and name

// In case of Stale Object excess property is allowed

//But if we write it as it will generate error

myType = { id: 2,  name: "Tom", age: 22 };

// But if we assign it as we are directly assiging a value

myType = { id: 2,  name: "Tom", age: 22 };

// Whereas in case of the myType = myType4; we assigning a object to an object

Here we should know the concept of Stale and Fresh.