Object in TypeScript

An object is a fundamental data type that represents a collection of key-value pairs, where keys are strings or symbols, and values can be of any data type, including other objects. Objects in TypeScript are similar to objects in JavaScript, but TypeScript adds static typing to enhance code quality and maintainability.

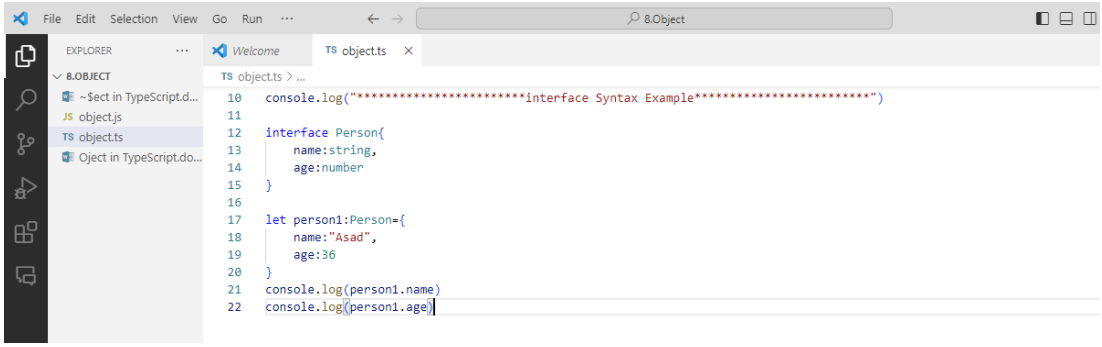
There are different ways to define objects in TypeScript:

Object Literal Syntax:

The Object Literal Syntax in TypeScript allows you to define and create objects using a concise and straightforward syntax. It involves specifying key-value pairs within curly braces {}, where keys are typically strings or symbols, and values can be of any data type, including other objects.

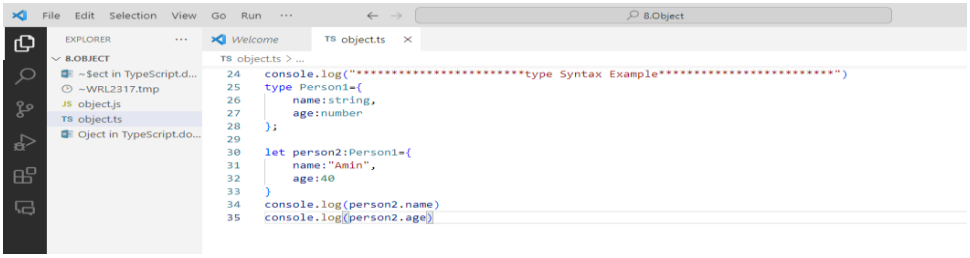
Interface:

You can use an interface to define the shape of an object and then create objects that adhere to that interface.



Type:

Similar to interfaces, you can use the type keyword to define the shape of an object, but assignment operator additionally used before start of curly braces.



What is a nested object?

Imagine you have an object, which is like a box that can hold information. Now, inside that box, you can have another smaller box. That smaller box is also an object. So, a nested object is basically an object that lives inside another object.

Advantage of nested object:

Let's say you want to organize information in a structured way. Instead of having everything in one big box, you can have smaller boxes (nested objects) inside the main box. This helps you represent information in a more organized and hierarchical manner, like building a tree of information

In TypeScript, a nested object refers to an object that is a property of another object. This means that the value associated with a certain key in an object is itself another object. This allows you to represent hierarchical or structured data in your TypeScript code.

```typescript
console.log("************************Example of Nested object in Typescript*************************");

type Author = {
    firstName: string;
    lastName: string;
};

type Book ={
    author:Author;
    name:string;
}
const myBook : Book ={
    author:{
        firstName:"Abdul",
        lastName:"Nasir"
    },
    name:"My Best Book"
}
console.log(myBook.name)
console.log(myBook.author)
```

```typescript
console.log("************************2nd Example of Nested object in Typescript*************************");
type Student={
    name:string,
    age:number,
    adress:{
        street:string,
        city:string,
        postalCode:number

    }
}
let student:Student={
    name:"Abdul Nasir",
    age:36,
    adress:{
        street:"No 1 ",
        city:"Karachi",
        postalCode:23520
    }

}

console.log(student.name)
console.log(student.age)
console.log(student.adress)
```

Intersection Types:

You can use intersection types (&) to combine multiple object types into a single type that has all the properties of each constituent type. This allows you to create more specific and complex types.