## Class & Interface

How do we write object orientated code in ES6?

**Object Orientation in JavaScript**

JavaScript has a prototype-based, object-oriented programming model. It creates objects using other objects as blueprints and to implement inheritance it manipulates what’s called a *prototype chain*.

We normally call the way object orientation is implemented in C++ or Java as the *Classical OO Pattern* and the way it’s implemented in JavaScript as the *Prototype Pattern*.

Although the prototype pattern is a valid way to implement object orientation it can be confusing for newer javascript developers or developers used to the classical pattern.

So in ES6 we have an alternative syntax, one that closer matches the classical object orientated pattern as is seen in other languages.

Under the hood the new syntax still uses the prototype pattern with constructor

functions and the prototype-chain. However, it provides a more common and convenient syntax with less boilerplate code.

TypeScript supports the ES6 class syntax but also adds some other feature like

access modifiers and interfaces, so in this lecture we’ll be writing TypeScript rather than pure ES6.

**Class**

A class is a blueprint for creating objects with specific functions and properties already attached to it, lets go through a simple example line by line:

class Person {

①

firstName = "";

②

lastName = "";

constructor(firstName, lastName) {

③

this.firstName = firstName;

this.lastName = lastName;

}

name() {

④

return `${this.firstName} ${this.lastName}`;

}

whoAreYou() {

return `Hi i'm ${this.name()}`;

⑤

}

}

1. We define a class using then class keyword.
2. We describe the properties we want on our class instance.
3. Each class has a special constructor function, this is called when we create an instance of a class with new
4. We describe the functions, also known as methods, that we want on our class instance.

⑤this in a method points to the class instance, the object that is created using this class.

### Class Instance

A class is a *blueprint* for creating an object, we call that created object an *instance of a class*, or a *class instance* or just *instance* for short.

We instantiate a class by using the new keyword and when that happens javascript calls the constructor function. We can pass to the constructer arguments which it uses to initialise properties or call other function, like so:

let asim = new Person("Asim","Hussain");

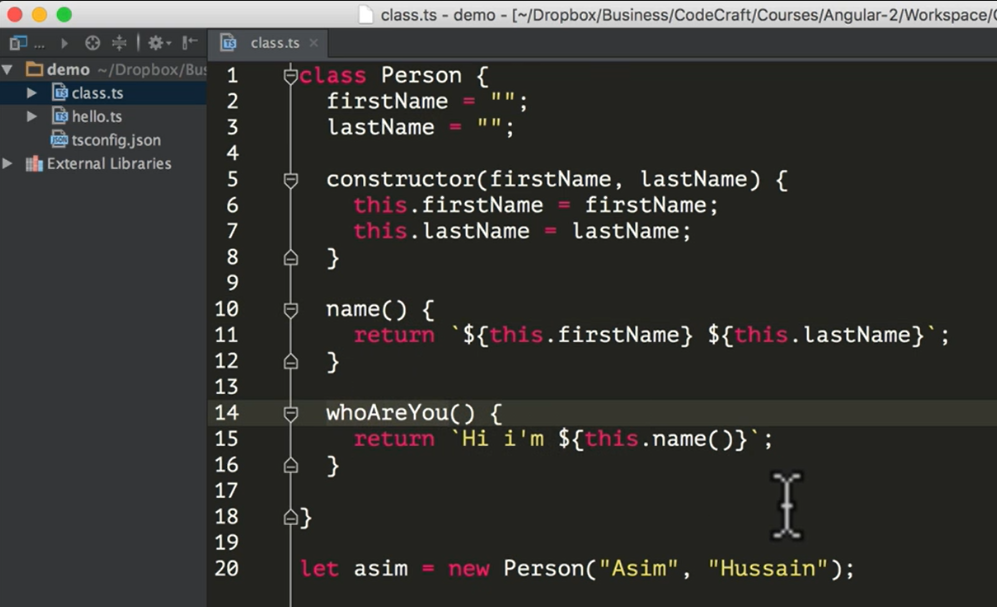
The above creates an instance of the *Person* class called *asim*.

The *asim* instance has the same properties and functions that are described on the Person class:

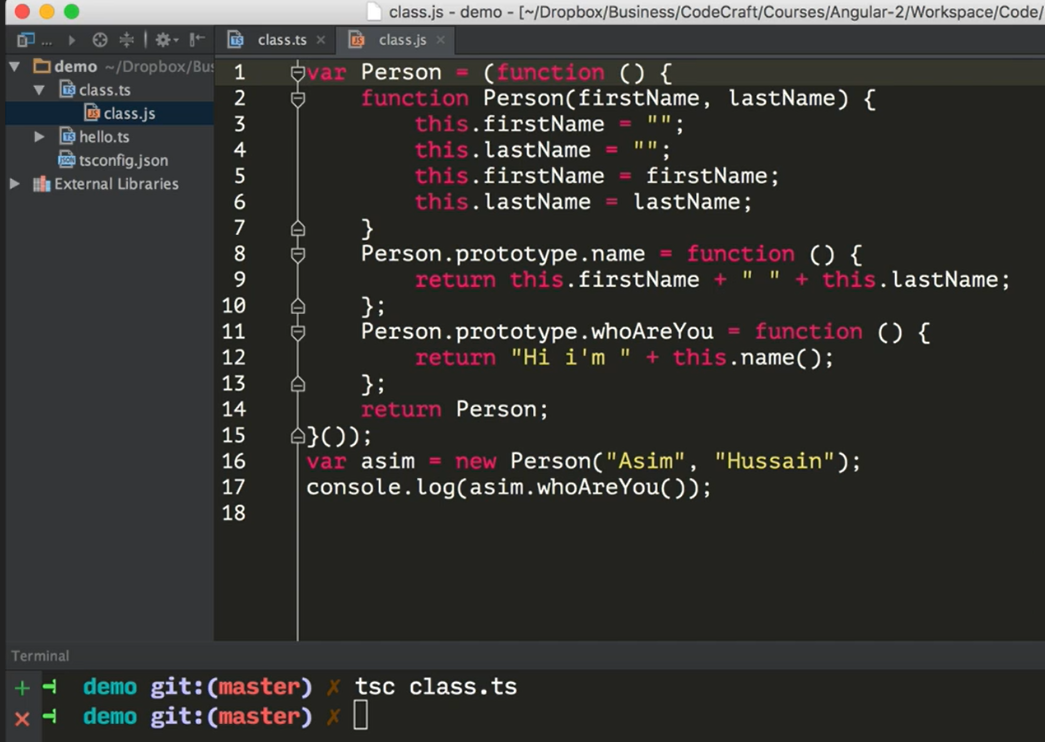
let asim = new Person("Asim","Hussain");

asim.whoAreYou()

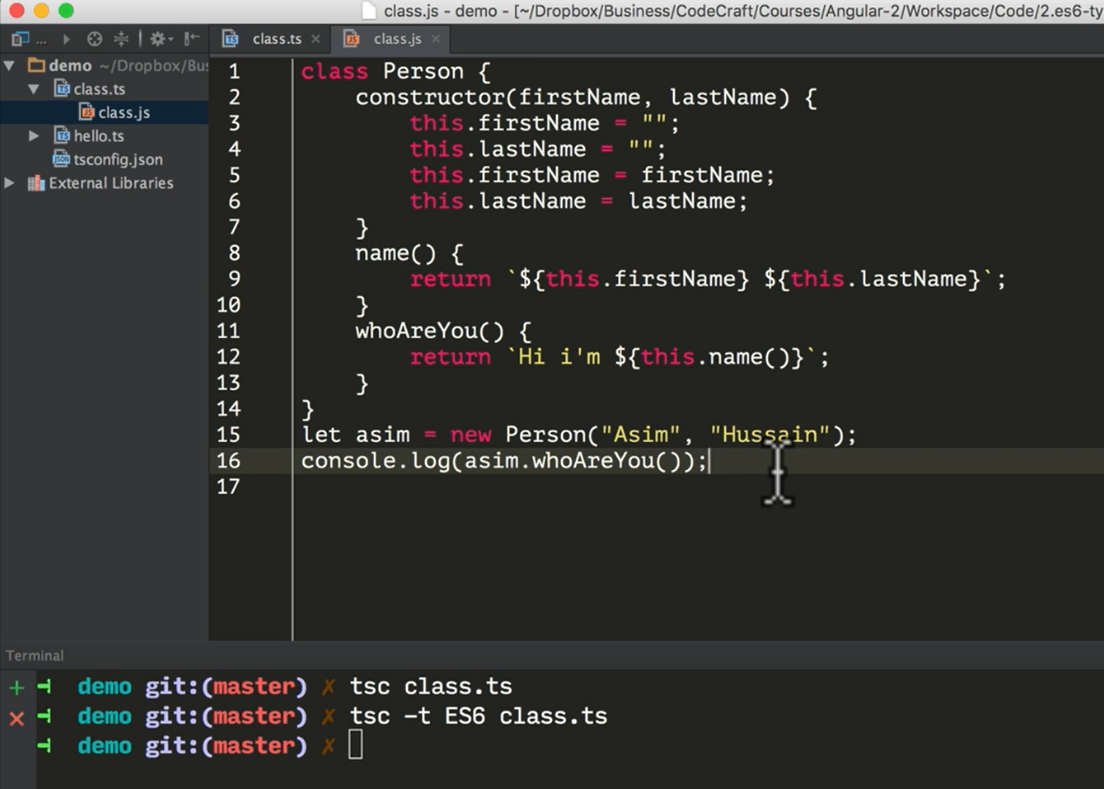
// "Hi i'm Asim Hussain"



Compiling into ES5, as you know that ES5 doesn’t support the class keyword



Compiling into ES6 which has **class** keyword



## Inheritance

A class can inherit from another class. We can create a class blue-print that *extends* an existing class blue-print by adding other methods or properties.

We do this by using the

extends

keyword, like so:

class Student extends Person {

①

course;

②

constructor(firstName, lastName, course) {

super(firstName, lastName);

③

this.course = course;

}

whoAreYou() {

④

return `${super.whoAreYou()} and i'm studying ${this.course}`;

⑤

}

}

1. We use the extends keyword to signal that this class inherits all the properties and methods from the *parent* Person class.
2. We can describe additional properties.
3. We use the super function to call the constructor of the parent class
4. We can *override* member functions of the parent class with our own versions.
5. In member functions super refers to the parent instance.

We can then instantiate this derived class like so:

let asim = new Student("Asim", "Hussain", "Angular 2");

console.log(asim.whoAreYou()); // Hi i'm Asim Hussain and i'm studying Angular 2

**Access Modifiers**

Everything we have learned so far about classes is pure ES6 JavaScript.

However TypeScript adds some nice functionality on top of ES6 classes, namely function and property *visibility* via *access modifiers*.

For example we can define the properties of our Person class as private, like so:

class Person {

private firstName = "";

private lastName = "";

constructor(firstName, lastName) {

this.firstName = firstName;

this.lastName = lastName;

}

}

And we create a function on our Student class called test() which tries to access one of these properties, like so:

class Student extends Person {

.

.

.

test() {

console.log(this.firstName);

}

}

And we tried to call this function from our Student instance, like so:

let asim = new Student("Asim", "Hussain", "Angular 2");

console.log(asim.test());

Compiling the above with typescript prints out this error:

error TS2341: Property 'firstName' is private and only accessible within class

'Person'.

By marking the firstName property as private it is now only visible from one of the methods on Person class.

We can also define *class methods* as private with the same effect. If we tried to call a private method from *outside* of a Person class, the typescript transpiler throws an error.

There are 3 access modifiers:

**public**

This is the default and means its visible everywhere.

**private**

Only member functions of the class it’s declared in can see this.

**protected**

Only the class it’s declared in and any class that *inherits* from that class can see this.

**Constructor shortcut**

A really common pattern in constructors is to use them to initialise properties via arguments you pass into the constructor, like in our example:

class Person {

private firstName = "";

private lastName = "";

constructor(firstName, lastName) {

this.firstName = firstName;

this.lastName = lastName;

}

}

As long as you are using an *access modifier* TypeScript lets us shorten this to:

class Person {

constructor(private firstName, private lastName) {

}

}

**Interfaces**

TypeScript has another feature called an interface. An interface can be used in a number of scenarios but by far the most common is when used with classes.

When used with classes the syntax looks like this:

class Person implements Human {

}

Human in the example above is an *interface*. An interface lets you describe the *minimum* set of **public** facing properties or methods that a class has.

Another way interfaces are explained is that they describe a set of rules the class has to follow, a *contract* it has to adhere to.

So for us a Human interface might look like:

interface Human {

firstName: string;

lastName: string;

}



Since interfaces are all about the

**public**

interface of a class they can’t have access

modifiers, the properties above have to be

*public*

.

If the Person class then doesn’t implement at least a firstName and a lastName then typescript throws an error like so:

error TS2420: Class 'Person' incorrectly implements interface 'Human'. Property 'firstName' is missing in type 'Person'.

Sometimes however we might want an interface to describe an optional contract. We can append ?

to the *name* of the property or function to mark it as *optional*, like so:

interface Human {

firstName: string;

lastName: string;

name?: Function;

isLate?(time: Date): Function;

}

**Summary**

In ES6 we now have a new way of writing object oriented code with the class syntax.

We can inherit methods and properties of one class into another by using the extends keyword.

Under the hood we are still using prototype based inheritance but the syntax is easier to understand and more familiar for developers who are coming from other languages.

TypeScript adds some extra functionality on-top of ES6 classes such as *access modifiers* and *interfaces*

**Listing**

[https://github.com/codecraftpro/angular2-sample-code/blob/master/2.es6-typescript/10.classinterface/script.ts](https://github.com/codecraftpro/angular2-sample-code/blob/master/2.es6-typescript/10.class-interface/script.ts)

*script.ts*

interface Human {

firstName: string;

lastName: string;

name?: Function;

isLate?(time: Date): Function;

}

class Person implements Human {

constructor(public firstName, public lastName) {

}

public name() {

return `${this.firstName} ${this.lastName}`;

}

protected whoAreYou() {

return `Hi i'm ${this.name()}`;

}

}

class Student extends Person {

constructor(public firstName, public lastName, public course) {

super(firstName, lastName);

}

whoAreYou() {

return `${super.whoAreYou()} and i'm studying ${this.course}`;

}

}

let asim = new Student("Asim", "Hussain", "typescript");

console.log(asim.whoAreYou());