Training TypeScript
Module: Types vs Interfaces

Peter Kassenaar

info@kassenaar.com

“What should I be using?”

The ongoing debate of `type` vs `interface`

What have we got?

- In TypeScript, we have the keyword `type` as well as `interface`
- `type`: use the equal sign (`=`)
- `interface`: use direct curly brace notation

```typescript
// A type with some properties
type Person = {
    name: string;
    age: number;
}

// Same properties, now in an interface.
interface IPerson {
    name: string;
    age: number;
}
```

?? Any difference ??

Example code: `../40-types-vs-interface.ts`

Short answer

There is no functional difference.

Use whatever floats your boat. In the compiled JavaScript both are gone!

TypeScript playground

v4.8.2 Run Export Share

```typescript
type Person = {
    name: string;
    age: number;
}

// Same properties, now in an interface.
interface IPerson {
    name: string;
    age: number;
}

// some variables, based on that type/interface
const person1: Person = {
    name: 'Peter',
    age: 10
}

const person2: IPerson = {
    name: 'Sandra',
    age: 20
}
```

.JS .D.TS Errors Logs Plugins

```typescript
"use strict";
// some variables, based on that type/interface
const person1 = {
    name: 'Peter',
    age: 10
};
const person2 = {
    name: 'Sandra',
    age: 20
};
console.log('I\'m a person with a type: ', person1);
console.log('I\'m a person with an interface: ', person2)
console.log('There is no difference....');
```

Works as expected

```
// some variables, based on that type/interface
const person1: Person = {
    name: 'Peter',
    age:  '10'
}
```

TS2322: Type 'string' is not assignable to type 'number'.

40-types-vs-interface.ts(12, 9): The expected type comes from property 'age' which is declared here on type 'Person'

Suppress with @ts-ignore Alt+Shift+Enter More actions... Alt+Enter

age: number

```
// some variables, based on that type
const person1: Person = {
    name: 'Peter',
    ag2e:  10
}
```

TypeScript is clever. Gives you some tips.

TS2322: Type '{ name: string; ag2e: number; }' is not assignable to type 'Person'.
Object literal may only specify known properties, but 'ag2e' does not exist in type 'Person'. Did you mean to write 'age'?

Suppress with @ts-ignore Alt+Shift+Enter More actions... Alt+Enter

ag2e: number

Classes can implement both

```
// 1d. Classes can implement both a type and an interface. A
class Peter implements Person{
    name: string;
    age: number;
}

class Sandra implements IPerson{
    name: string;
    age:number;
}

const peter = new Peter();
peter.
con
rfaces
```

name	string
age	number

So?

Use your *personal preference*, or company standard.

But:

Under the covers there are some differences in how TypeScript treats both.

Interfaces

Unique features of interfaces

Unique features of interfaces

- 1. Interfaces can extend other interfaces.
- With a type this is not possible

```
interface IPerson {
    name: string;
    age: number;
}
```

```
interface Peter extends IPerson{
    teacher: boolean;
}

const me: Peter ={
    name: 'Peter',
    age: 10,
    teacher: true
}
```

2. Declaration merging

- This is valid ('Define an interface multiple times')
- The resulting object needs to have the props of *both* interfaces

```
interface IPerson {
    name: string;
    age: number;
}
…
```

```
interface IPerson{
    city: string;
}
```

```
const harry: IPerson={
    nam
    age
}
// 2c.
```

TS2741: Property 'city' is missing in type '{ name: string; age: number; }' but required in type 'IPerson'.

40-types-vs-interface.ts(74, 9): 'city' is declared here.

Remove unused constant 'harry' Alt+Shift+Enter More actions... Alt+Enter

```
const types_vs_interfaces.harry: IPerson
```

src/ts/40-types-vs-interface.ts

3. Interfaces are geared to…

- Interfaces are more geared towards `objects`, `functions` and `classes`.
- They generally are more used in a `OOP-style` of programming.
- Types are more used in a functional way of programming, composing complex types from simple types

"Composition (types) over inheritance (interface)"

So: Your choice

Types

Types are much like interfaces nowadays

Types are an alias for a shape of data

```
type Teacher = boolean;

// 3b, now you can use the type as an alias
const peter: Teacher = true; // valid
```

```
// A type is therefore also called a "type alias".
type Teacher = boolean;

// 3b, now you can use the type as an alias
const peter: Teacher = 'true'; // <== error, wrong type (st
}
```

TS2322: Type 'string' is not assignable to type 'boolean'.

Remove unused constant 'peter' Alt+Shift+Enter More actions... Alt+Enter

```
const types_vs_interfaces.peter: boolean
```

3b, now you can use the type as an alias

Types have Intersections and Unions

- Types cannot extend other types, as interfaces can
- But, types have subtypes, intersections and unions
  - More or less the same outcome, but different syntax.

```
// 3d. Intersection Types. Use the ampersand:
type tStudent = {
    student: boolean
}

// type tPerson is *intersected* with type tStudent
type tPerson ={
    name: string;
    age: number;
} & tStudent

// we need to use all the properties here, b/c of intersection type
const sandra : tPerson={
    name: 'Sandra',
    age: 20,
    student: true
}
```

Intersection Types

- Use the ampersand, pronounced as AND

```
// 3e. Of course you can create additional types like so:
type Sandra = tPerson & Student;
```

```
const mySandra: Sandra = {
    name: 'Sandra',
    age: 20,
    // student: true // <== ERROR when omitted.
}
```

This can come in handy if you have data coming in from multiple endpoints and want to combine (‘compose’) it in one frontend type, but retain type safety

Union Types

- A type is of one type OR the other
- The pipe symbol | is pronounced OR
- They can also be merged together (using ALL the properties of given types)

```
type Sandra = Person | Student;
const mySandra: Sandra ={
    student:true
}
```

So, unique features of types :

- Types are aliases for the shape of the data. They are static.
- Types can be intersected (&) or united (|)
- Sometimes easier to use than interfaces
  - In a composition/functional programming style environment
  - Personal preference!

- So again, types and interfaces are MOSTLY the SAME

Types and interfaces “In the wild”:

Using a lot of objects and/or classes? ➔ Use `interfaces`

Not? ➔ use `types`

But then again, it mostly doesn’t matter!

However: be consistent with

Yourself

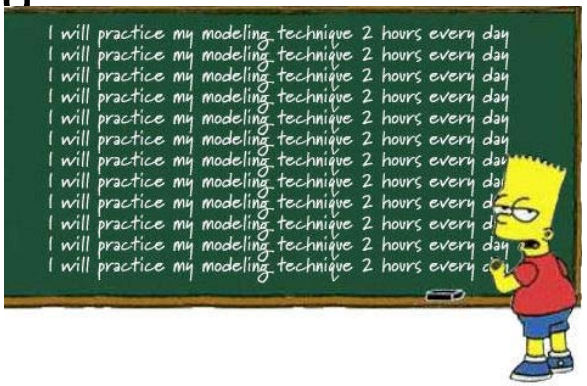
Your team

Your company


Demo

Workshop

- Create a `Car type` or make up a `type` yourself. Give it some properties.
  - Create a variable of that type
- Also create an `ICar interface` (or again, your custom type).
  - Also create a variable based on that interface
- Log both variables to the console
- Practice with `interface extending` as explained in the slides
- Practice with type `intersection` and `union` as explained in the slides



- Example code: `../40-types-vs-interface.ts`