TypeScript Lab

Let's exercise some TypeScript! We're going to ask you to create some TS types that'll be used in future labs. We're hoping you get a feel for the new demands that TS puts on you -- demands of safety, demands of precision. And while many people welcome those demands, others will resent them. So we warn you; this lab may be painful if you're in that latter category.

We're hoping that this experience will make you open to a sane balance between the speed & freedom of dynamic JavaScript vs the safety & protections of typed TypeScript. Some hints:

1. Read the entire lab before you begin. There are comments and suggestions offered after-the-fact that can help you to solve problems.
2. When creating types, you can create them as classes, interfaces, or types. Each have their own advantages. You have to decide which is best.
3. Class properties and method variables can be strongly typed. But the types can be omitted when it makes sense to. (This is hotly debated topic).
4. If you're finding TypeScript is too picky, remember that the tsconfig file has a "strict" flag that can be set to false. This is your way of saying "TypeScript, please chill for a minute".

With that said ... Get after it!

Making our types

We're going to be working with products, orders, and customers so it might be safer to create types to define the shape of the objects we'll be working with. We'll start by putting them in a shared folder.

1. Create a new folder under *src/app* called the *shared* folder.
2. In it, create five new files called customer.ts, location.ts, order.ts, orderLine.ts, and product.ts.
3. These will be our type files. In each, create one class, interface or type. You get to decide which one you want for each. Give them the following properties:

Customer

id	number
givenName	string
familyName	string
companyName	string
address	string
city	string
region	string
postalCode	string
country	string
phone	string
email	string
imageUrl	string
password	string

Order

id	number
customerID	number
status	number
orderDate	Date
shipVia	number
shipping	number
tax	number
shipName	string
shipAddress	string
shipCity	string
shipRegion	string
shipPostalCode	string
shipCountry	string
lines	Array<OrderLine>
customer	Customer

Product

id	number
name	string
description	string
price	number
imageUrl	string
featured	boolean

OrderLine

quantity	number
productID	number
locationID	string
price	number
picked	boolean
product	Product
location	Location

Location

id	string
description	string
productID	number
quantity	number
product	Product

Note that Order uses OrderLine and Customer. Don't forget that you'll need to import these classes into Order.ts for this to work. This is true for Location and OrderLine as well.

Using the Order type in a component

The ShipOrderComponent's eventual purpose is to get an order ready to ship to the customer. So obviously it needs an order!

4. Edit shipping/ship-order.component.ts. Give the class a public property called *order* that is of type Order. (Yes, the one you just created). Don't forget you'll need to import it from Order.ts at the top.*
5. You should get an error at this point because the order property of the ShipOrderComponent class might be nullish and TS doesn't like properties accidentally becoming nullish. Discuss with your partner some ways that you might solve that.

One way to solve the problem is to make Order a **type** and make all its properties optional with the "?". But if you do that, you'll have to handle the possibly null properties throughout the html template. That's a viable alternative, but in this situation, there's a cleaner way with a lot less coding.

6. Make Order a **class** instead of a type. Then, declare all its properties with the exclamation point like so:

```
export class Order {
  id!: number;          // <-- Note the exclamation point
  customerID!: string;  // <-- Note the exclamation point
  etc. etc.
```

This says that the properties will exist, but it's okay for them to be null.

7. Now, back in ship-order.component.ts, instantiate an Order as soon as you declare it. You can do it like this:

```
order: Order = new Order();
```

8. We've provided a starter to create a fake order so you don't have to type every single property. Go look in /starters/codeSnippets/anOrderReadyToShip.js. Copy its contents into the ngOnInit() method of ship-order.component.ts.

Unless you wrote your types in a very particular way, you're encountering many compile errors. Fix those by making properties optional with "?" or "!". Or you could just make it really easy and use the "any" type.

Let's take a breather for a second. If you were using plain JavaScript or set "strict": false in tsconfig.json or if you took the easy way out and made everything an "any" type, you wouldn't have these problems. It's a lot simpler to code that way, but it's also more dangerous at runtime. You and your team will have to decide standards on how to handle these situations - quicker coding or safer at runtime?

---

* We're only going to remind you to add imports statements occasionally from now on so don't forget going forward.

Hey, don't be discouraged. You're not alone. TypeScript makes learning Angular more complicated. Once you get through the learning curve, you'll be writing safer and better code!

Let's do one that is a little easier.

9. Add this to dashboard.component.ts and orders-to-ship.component.ts

```
orders: Order[] = new Array<Order>();
```

This means that we need a list of Orders. Only an Order can be added to this list. We're also initializing it so it's not nullish.

10. And now let's do last one. If you look in the starters/codeSnippets folder, you'll see a file called SomeOrders.js. Open the file and copy its contents to the ngOnInit() method of the DashBoardComponent. Be ready to adjust the data or your business classes.
11. This will populate a property called "orders". Do a console.log() in ngOnInit() to make sure it is reading alright. We'll use this when we talk about directives soon. Stand by for that!

Bonus!! Exploring tree-shaking

If you're finished early and have extra time, let's explore how webpack and TypeScript interact to make your payloads smaller.

12. Open a command window and compile your site by typing in
`ng build`
or
`ng build --configuration=development`
13. Find the dist directory and look in there. You should see a main.*.js file. This is the file that will be served once we go live.
14. Go ahead and edit main.*.js. You and your partner look around in there. Do you see your ShipOrder component in there? _____ (You shouldn't unless you included it in app.component.html). How about your TypeScript classes? Are some there but not others? _____ Discuss with some of the students around you why this makes sense.

What's going on is this: TypeScript files are only included if they're used. If not, they're excluded from the bundle, keeping the size smaller.