Composition with Components

The spirit of web components is to allow encapsulation so that our site is easier to maintain and easier to extend. And we have the opportunity to do that right now. If you look at the main page you'll see a link for "Ship Orders" but on our main page, we already have a list of orders to ship. Aha! We have an opportunity for re-use!

1. Open a command window and use the Angular CLI to generate a new component called ListOfOrders. It should exist in the shipping folder. (Hint: use the --flat flag and specify the folder as part of the name ie. shipping/ListOfOrders)
2. Edit Dashboard.component.html and add a tag to show the list-of-orders component. Maybe put it just above the list-of-orders <div>.
3. Run and test. If you can see your new component hosted inside the dashboard, you've just implemented composition.
4. Now edit dashboard.component.html and find the html that creates the list of orders. Cut it from there and paste it into your new list-of-orders.component.html.
5. You'll see various compile errors because properties and methods in the HTML are not in the list-of-orders.component.ts TypeScript class.
6. Fix those errors by copying/cutting from dashboard.component.ts and pasting them into list-of-orders.component.ts.
7. Run and test. You won't see any orders lists because the "orders" property is defined in DashboardComponent but is trying to be displayed in ListOfOrders. We need to pass those orders into the ListOfOrders component. Let's do that in the next few steps. Read on!

Property binding between components

8. Open ListOfOrders and create a new property called orders. Mark it with an "@Input" annotation. (Hint: In order for this to work, you're going to have to import *Input* from @angular/core).
9. Now to pass it in the HTML. Open dashboard.component.html. Find where you've included the tag to list-of-orders. Add this property binding to it:

```
[orders]="orders"
```

   This says to take the orders property from the host component and pass it down into the inner component as orders. (They happen to have the same name but they could be different if you prefer).

10. Run and test again. You should see the list in your dashboard again.

Cool. What we've done will make our app more modular and easier to understand. But we now have an opportunity for re-use.

Reusing a component

Notice that the AppComponent has a menu choice at the top called "Ship Order". This component will present a list of orders that are ready to ship and then allow the user to click on one to see the details for that one order. Hey, that's like ListOfOrders! Let's reuse it.

11. Edit orders-to-ship.component.html and add a tag to display the list-of-orders component. (Hint: Don't forget your property binding.). Remove the hardcoded order-list <div>.

12. Look in the DashboardComponent.ts in the ngOnInit method and copy the creation of this.orders to OrdersToShipComponent.ts.
13. Run and test. Once the list of orders is available, you've successfully used and re-used a component.

14. Bonus! Remember that you had previously created some styles for that list. They're in dashboard.component.css. If you move those styles to list-of-orders.component.css, they'll apply on both Dashboard and OrdersToShip. Go ahead and move them.

Extracting the shipping label

Take a look at an order in the browser. Remember that nifty shipping label at the bottom? That should probably be relocated to a new component. Let's extract it as well but this time you'll do it with fewer instructions.

15. Create a new component called shipping/ShippingLabel.
16. Include it at the bottom of ship-order.component.html. Make sure you pass the order as a property binding.
17. Move the shipping label's HTML into shipping-label.component.html.
18. Move the shipping label's CSS into shipping-label.component.css.
19. Edit shipping-label.component.ts and add the order as a property. Mark it with @Input.

You'll know you have the refactor correct when you can see the shipping label again.

Got it? Way to go!