# Angular Best Friends

## Module 3 Exercise 1 – Input and output

### Outcome

The goal of this exercise is to get familiar with input and output properties. Using this properties is a fundamental technique used to pass data between components. In this exercise we’ll basically create a new very basic Angular app and we’ll pass data between components.

### Steps

1. Create a folder where you’ll put the application
2. Having NodeJs and Angular CLI install open a command prompt, PowerShell or the Terminal in Visual Studio code.
3. Create a new Angular application

**Command:**

**ng new input-output-demo**

1. Change directory to “input-output-demo”

**Command**

**cd input-output-demo**

1. Install the angular/devkit-core npm module

**Command:**

**npm install @angular-devkit/core**

1. Open the folder in VS Code (if you ran the commands from VS Code then you are already there 😊)
2. Create a module called “customer”

**Command**

**ng g m customer**

1. Create a customer component for the new module

**Command**

**ng g c customer**

1. Note that since we have named the component the same way we named the module, the Angular CLI has created the component in the right place and has also declared our new component in the Customer Module.
2. Create a new child component

**Command:**

**ng g c customer/customer-details**

1. Go to the customer.component.html file delete what’s there and replace it with the following code

<h1>Customer</h1>

<br />

<app-customer-details [customer]="customer"></app-customer-details>

1. Go to the root component “customer.component.ts”. Delete all content and replace it with the following code:

import { Component, OnInit } from '@angular/core';

@Component({

selector: 'app-customer',

templateUrl: './customer.component.html',

styleUrls: ['./customer.component.css']

})

export class CustomerComponent implements OnInit {

customer: any;

constructor() { }

ngOnInit() {

this.customer = {

name: "John Doe",

address: {

city: "Cluj-Napoca"

}

}

}

}

1. Here we have basically created a new “customer” property of type “any”. Then in ngOnInit() we have created a customer object holding a name and an address. The address is de facto an object holding a property called “city”.
2. Go to the “customer-details.component.ts” file. Delete all the code and replace it with the following snippet:

import { Component, OnInit, Input } from '@angular/core';

@Component({

selector: 'app-customer-details',

templateUrl: './customer-details.component.html',

styleUrls: ['./customer-details.component.css']

})

export class CustomerDetailsComponent implements OnInit {

@Input() customer:any;

constructor() { }

ngOnInit() {

}

}

1. What we did is simply added an input property called customer and of type any
2. Go to the “customer-details.component.html”. Delete all the code and replace it with the following snippet:

Name: {{customer.name}}

<br />

City: {{ customer.address.city}}

1. Now we have to wire everything up. To do this, go to the “app.module.ts” file. Add the “CustomerModule” in the “imports” array:

imports: [

BrowserModule,

CustomerModule

],

1. To get everything working we will also need to export the CustomerComponent in the CustomersModule. Go to the “customers.module.ts” and export the components:

@NgModule({

imports: [

CommonModule

],

exports: [

CustomerComponent

],

declarations: [CustomerComponent, CustomerDetailsComponent]

})

1. Now we have to display this at the root of our application, so on the homepage. Therefore, go to the “app.component.html” file delete all the content and replace it with the following:
2. Let’s run the application!

**Command**

**ng server --open**

Now it’s time to concentrate on output properties.

1. Go back to the “customer-details.component.ts” file. Add an output property by adding the following line just under your input property:

@Output() customerChanged = new EventEmitter<any>();

Also import “Output” and “EventEmitter”

import { Component, OnInit, Input, Output, EventEmitter } from '@angular/core';

1. We’ll now simulate a customer changed event. To do this go to the “customer-details.component.html” file and add a button there. Add a “click” event to it to which we hook up a function called “change()”. We’ll add the function in the next step. For now, here’s the button:
2. <br />
3. <button (click)="change()">Change customer</button>
4. Go to the component code in “customer-details.component.ts” and add the change function after the ngOnInit() function:
5. change() {
6. this.customer.name = "Jane Doe";
7. this.customerChanged.emit(this.customer);
8. }
9. We now emit an information that the customer has changed. Now on the container component we need to listen to this event and probably do something about that. So now we jump to the “customer.component.html” file and subscribe to the previously created output property on the child. So replace the child component selector with the following one:
10. <app-customer-details [customer]="customer" (customerChanged)="changed($event)"></app-customer-details>

1. Now go to the TS file “customer.component.ts” and create a function to do something when the event triggers. Place the function after the ngOnInit() function:

changed(customer: any) {

this.customer = customer;

}

1. Save everything. Go back to the browser where our application should still be open. If not, just run ng server –open once again. Now you should see a button and when you click on it the name of the customer should change.

How does this work? Well, when we click the button, in the presentation component we change the name of the customer and emit this change. The container component listen to these changes and it also updates the customer with what it has received from the child’s output property. But since the “customer” property is an input property for the child, the new customer name propagates back to the presentation component and hence we see the updated name.