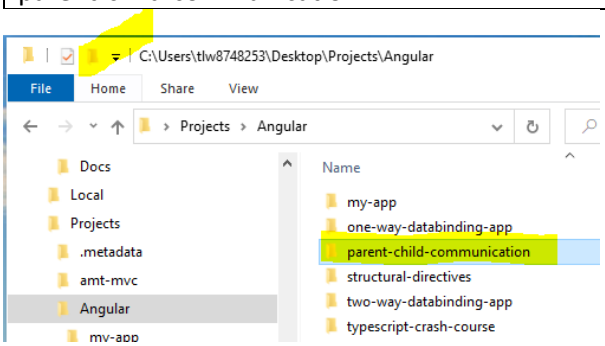
Table of Contents

Lecture and project for parent-child-communication

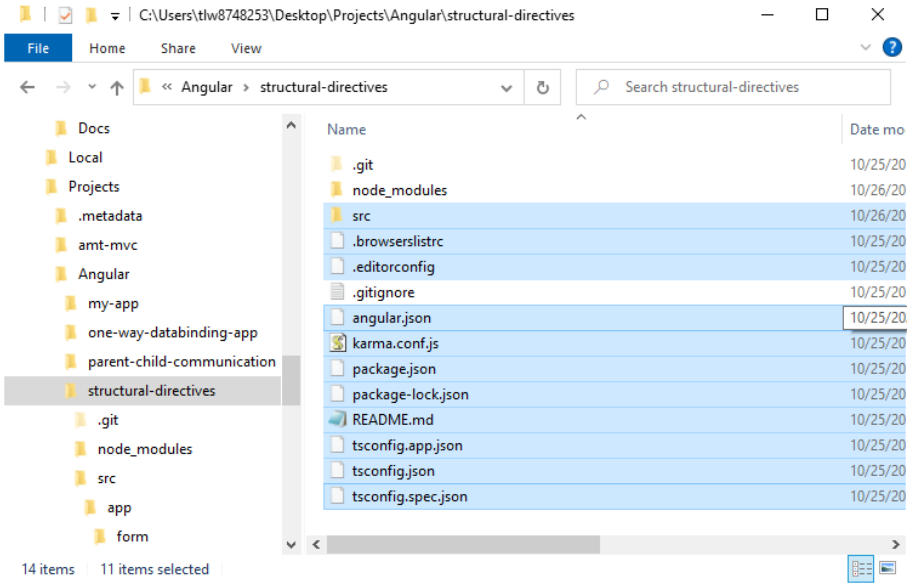
Copy previous project for this project

In your Angular project folder create a directory:

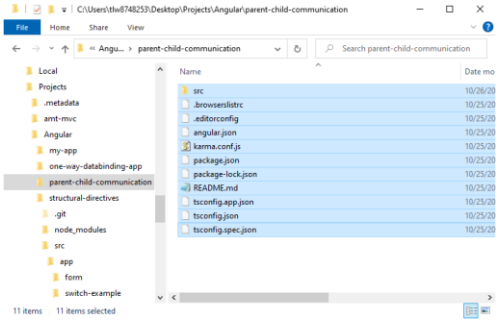
parent-child-communication



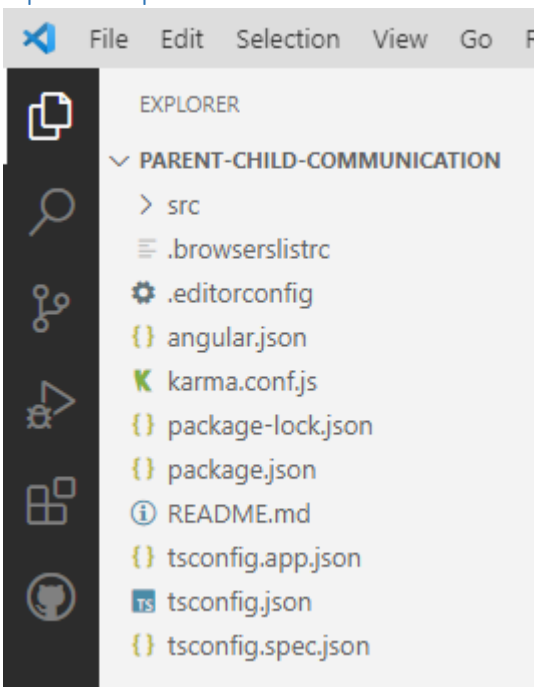
Copy all but the .git folder, node_modles folder, and .gitgnore file to the parent-child-communication.
To select multiple file hold the ctrl key while left clicking the mouse button.
Once all files are selected press ctrl key and press c key at the same time “ctrl+c”



Go into the parent-child-communication folder and paste the files “ctrl-v”:



Open the parent-child-communication folder in VS Code



Will need to update project files manually since it was copied and not created

File: package.json

Change:

```
"name": "structural-directives",
```

To:

```
"name": " parent-child-communication",
```

File: index.html

Change:

```
<title>StructuralDirectives</title>
```

To:

```
<title>ParentChildCommunication</title>
```

File: app.component.ts

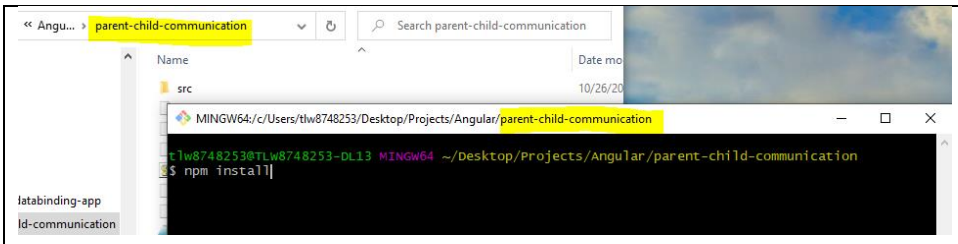
Remove the title:

```
title = 'structural-directives';
```

Next do a npm install for the Angular components and dependencies

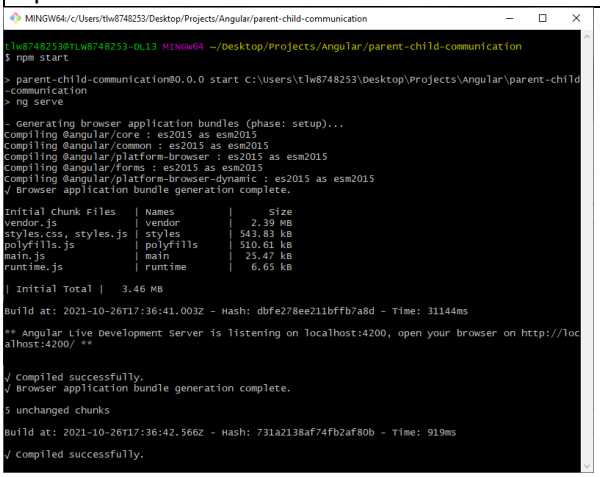
In a Git Bash window in the parent-child-communication folder type:

```
npm install
```



Start the application

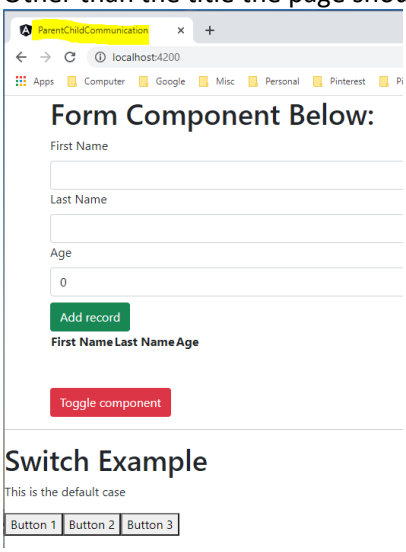
```
npm start
```



Open the application in the browser

```
http://localhost:4200/
```

Other than the title the page should look the same as the last project.

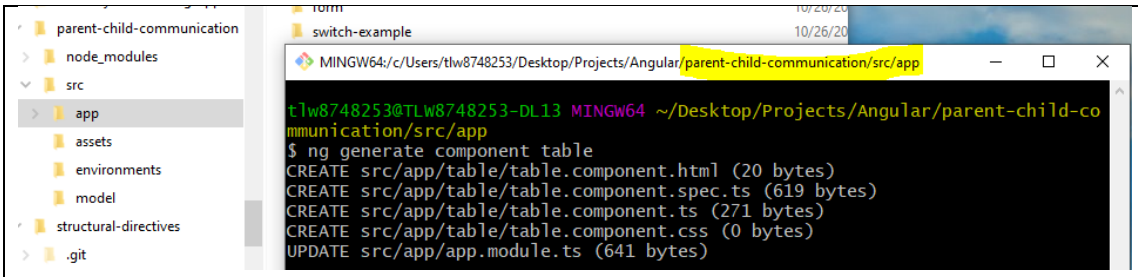


Separate out Person display table into a different component

Generate a new component

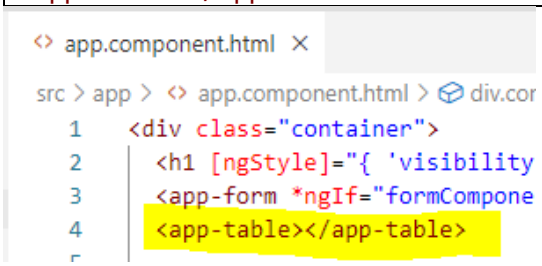
In a Git Bash window in the projects src/app folder create a new component

```
ng generate component table
```



Add the table component to the app.component.html file:

```
<app-table></app-table>
```



Move the display table from form.component.html the table html

Highlight the code then “ctrl-x” to cut move to table.component.html and replace the code there “ctrl-v”

```
<table>
    <thead>
      <tr>
        <th>First Name</th>
        <th>Last Name</th>
        <th>Age</th>
      </tr>
    </thead>
    <tbody *ngFor="let person of people">
      <tr>
        <td>{{ person.firstName }}</td>
        <td>{{ person.lastName }}</td>
        <td>{{ person.age }}</td>
      </tr>
    </tbody>
  </table>
```

app.component.html | form.component.html

src > app > form > form.component.html > table

```
<div>
    <label class="form-label">First Name</l
    <input [(ngModel)]="firstNameInputValue
</div>
<div>
    <label class="form-label">Last Name</la
    <input [(ngModel)]="lastNameInputValue"
</div>
<div>
    <label class="form-label">Age</label>
    <input [(ngModel)]="ageInputvalue" clas
</div>
<div>
    <button (click)="addRecord()" class="bt
</div>

<table>
    <thead>
        <tr>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Age</th>
        </tr>
    </thead>
    <tbody *ngFor="let person of people">
        <tr>
            <td>{{ person.firstName }}</td>
            <td>{{ person.lastName }}</td>
            <td>{{ person.age }}</td>
        </tr>
    </tbody>
</table>
```

app.component.html | form.component.html | table.component.html

src > app > table > table.component.html > table

```
<table>
    <thead>
      <tr>
        <th>First Name</th>
        <th>Last Name</th>
        <th>Age</th>
      </tr>
    </thead>
    <tbody *ngFor="let person of people">
      <tr>
        <td>{{ person.firstName }}</td>
        <td>{{ person.lastName }}</td>
        <td>{{ person.age }}</td>
      </tr>
    </tbody>
</table>
```

The webpage will fail to compile at this time. The people object does not exists.

```
Error: src/app/table/table.component.html:9:34 - error TS2339: Property 'people' does not exist on type 'TableComp
nent'.

9     <tbody *ngFor="let person of people">
                                   ~~~~~~

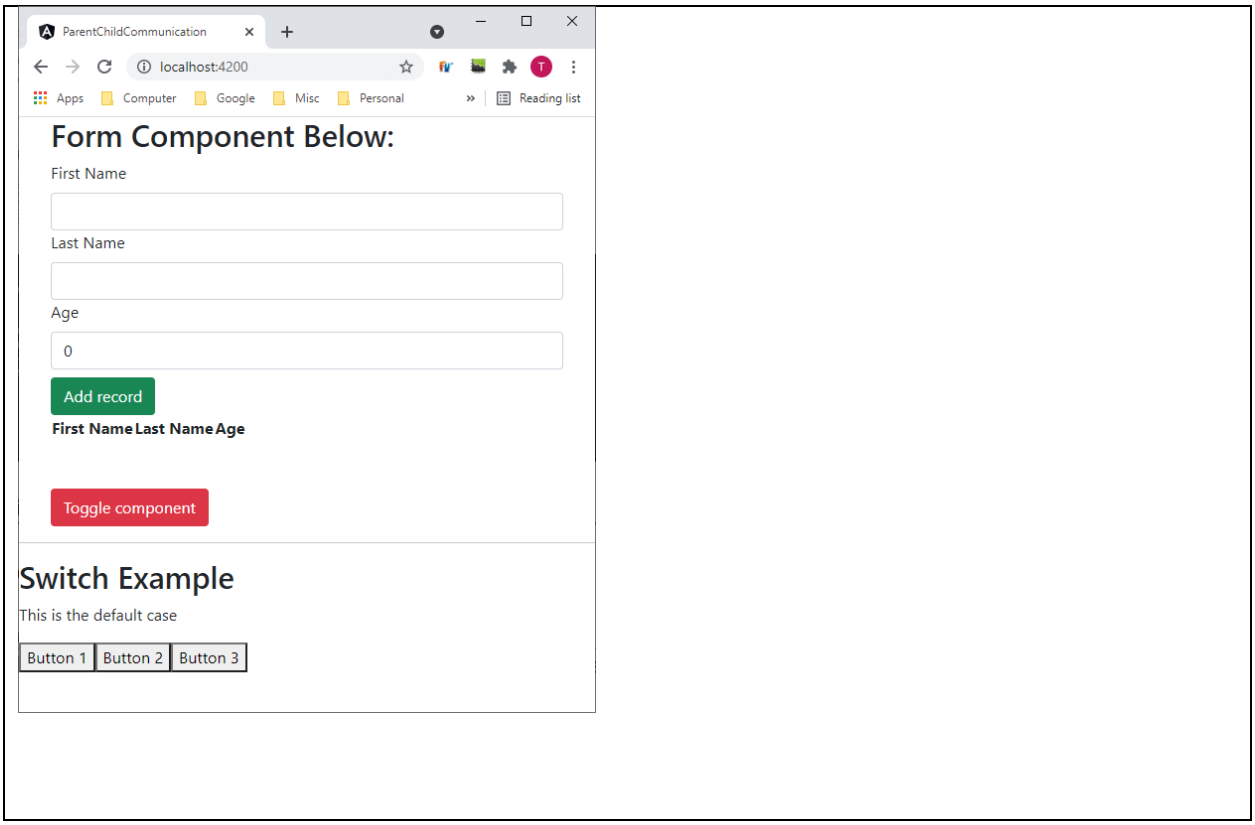
  src/app/table/table.component.ts:5:16
    5   templateUrl: './table.component.html',
                     ~~~~~~~~~~~~~~~~~~~~~~~~
    Error occurs in the template of component TableComponent.


✖ Failed to compile.
```

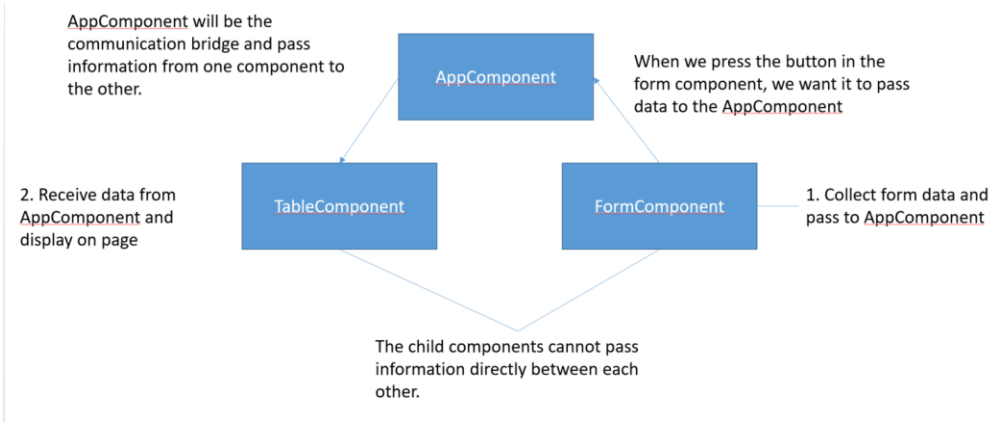
Import the person interface in the table.component.ts files

```
…
import { Person } from '../../model/person';
…
people: Person[] = [];
```

The changes will allow the code to compile but the mapping to display the data in the table is not finished.


ParentChildCommunication
localhost:4200
Apps
Computer
Google
Misc
Personal
Reading list
Form Component Below:
First Name
Last Name
Age
0
Add record
First Name Last Name Age
Toggle component
Switch Example
This is the default case
Button 1
Button 2
Button 3

Create a communication bridge between the children components



1. Collect form data and pass to AppComponent

Update app components

Update app.component.ts

```
…
import { Person } from '../model/person';
…
people: Person[] = [];
…
  onAddPerson(event: Person) {
    this.people.push(event); // This event object will be a Person object

    console.log(this.people);
  }
```

app.component.html ● TS app.component.ts ×

src > app > TS app.component.ts > …

```
1  import { Component } from '@angular/core';
2
3  import { Person } from '../model/person';
4
5  @Component({
6    selector: 'app-root',
7    templateUrl: './app.component.html',
8    styleUrls: ['./app.component.css']
9  })
10 export class AppComponent {
11
12   people: Person[] = [];
13
14   formComponentShouldBeDisplayed: boolean = true;
15
16   onToggleButtonClick() {
17     this.formComponentShouldBeDisplayed = !this.formComponentShouldBeDisplayed;
18   }
19
20   onAddPerson(event: Person) {
21     this.people.push(event); // This event object will be a Person object
22
23     console.log(this.people);
24   }
25
26 }
```

Update app.component.html

Change:

```
<app-form *ngIf="formComponentShouldBeDisplayed"></app-form>
```

To:

```
<app-form (addPerson)="onAddPerson($event)" *ngIf="formComponentShouldBeDisplayed"></app-form>
```

The above change is a listener to an event to be added to form components. It binds to the addPerson event that will be added.

Update form components

Update form.component.ts file

Remove the Person array from form.component.ts

```
people: Person[] = [];
```

Make additional changes to the form.component.ts file:

Note: the tag @Output is a decorator

Change:

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

To:

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

Add:

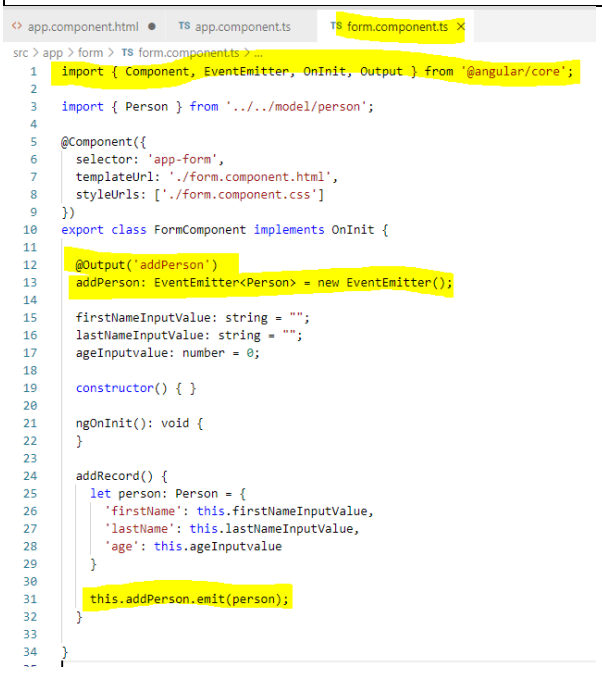
```
  @Output('addPerson')
  addPerson: EventEmitter<Person> = new EventEmitter();
```

Replace:

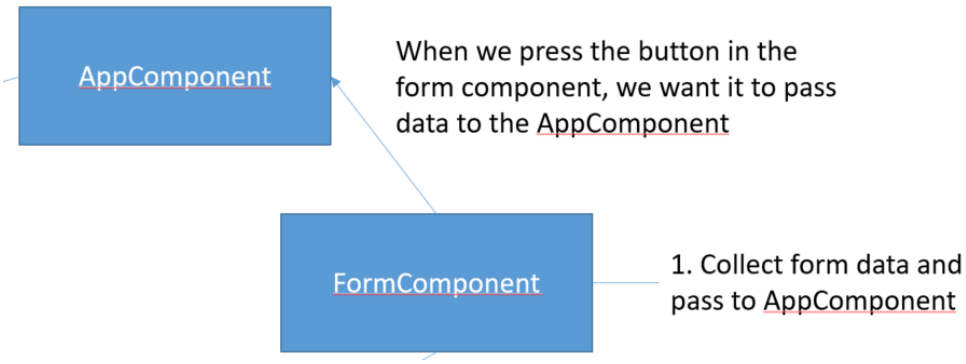
```
this.people.push(person);
```

With:

```
this.addPerson.emit(person);
```



The above changes complete half of the diagram:



2. Receive data from AppComponent and display on page

Update form components

Add input decorator to table.component.ts

Change:

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

To:

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

Add:

```
@Input('myPeople')
```

Change:

```
people: Person[] = [];
```

To:

```
myPeople: Person[] = [];
```

app.component.html | table.component.ts | app.component.ts

src > app > table > table.component.ts > ...

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

import { Person } from '../../model/person';

@Component({
  selector: 'app-table',
  templateUrl: './table.component.html',
  styleUrls: ['./table.component.css']
})
export class TableComponent implements OnInit {

  @Input('myPeople')
  myPeople: Person[] = [];

  constructor() { }

  ngOnInit(): void {
  }

}
```

Now add property binding for the above changes.

Update app.components.html

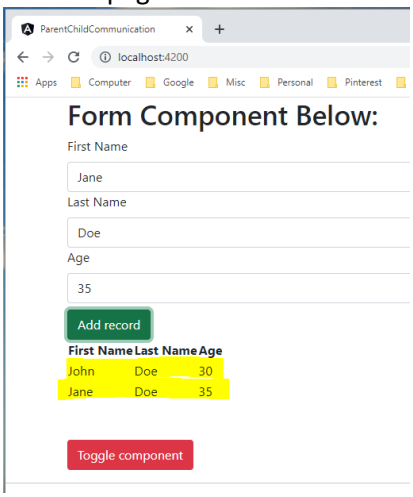
Change:

```
<app-table></app-table>
```
To:
```
<app-table [myPeople]="people"></app-table>
```

Update table.component.html

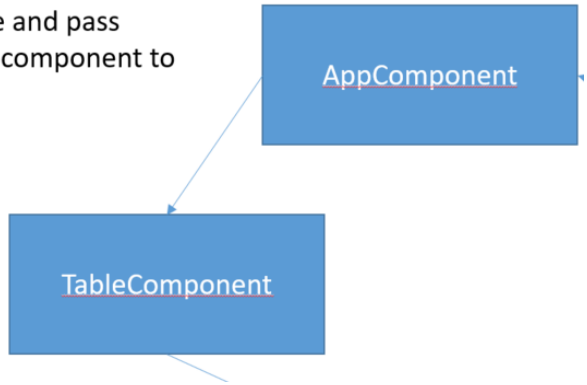
Change:
```
<tbody *ngFor="let person of people">
```
To:
```
<tbody *ngFor="let person of myPeople">
```

Test the page:



This finishes the application and the second part of the diagram.

AppComponent will be the communication bridge and pass information from one component to the other.



2. Receive data from AppComponent and display on page

Notes from this lecture and project is found in the appendix.
Appendix: angular-component-to-component-communication.md

Appendix: angular-component-to-component-communication.md

````
# Component to Component Communication

Before this point, we only discussed passing data from the template to the class of a single component. However, in an application with many different components all interacting with each other, we need a way to pass data from one component to another. We can break down this component to component communication in the following ways:
1. parent-to-child
2. child-to-parent

## Parent-to-child
If our parent component has some data that we would like to pass to the child, we would need to utilize the `@Input()` decorator in order to have a variable inside of our child component behave as an "attribute".

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

import { Person } from '../../model/person';

@Component({
  selector: 'app-table',
  templateUrl: './table.component.html',
  styleUrls: ['./table.component.css']
})
export class TableComponent implements OnInit {

  @Input('myPeople')
  myPeople: Person[] = [];

  constructor() { }

  ngOnInit(): void {
  }

}
```

From the perspective of the parent component, whenever we reference this TableComponent in the example above, there is an attribute of the `<app-table>` tag that is called myPeople. If we think back to the idea of one-way databinding, the way that we can bind information from a component to an element within that component's template is through property binding. So, we can perform property binding on this myPeople attribute as well.

So, inside of our app.component.html, where we are displaying the table component, we can bind a variable from our app component class to this attribute. This effectively binds the people variable that exists in the parent (app component) to the myPeople variable in the child component (table component)
```html
<app-table [myPeople]="people"></app-table>
```

## Child-to-parent
If our child component has some data that we would like to pass to a parent component, we would need to utilize the `@Output()` decorator in order to emit an event whose event object contains that data.

In our child component's class file:

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

import { Person } from '../../model/person';

@Component({
  selector: 'app-form',
  templateUrl: './form.component.html',
  styleUrls: ['./form.component.css']
````
````

```
})
export class FormComponent implements OnInit {

  @Output('addPerson')
  addPerson: EventEmitter<Person> = new EventEmitter();

  firstNameInputValue: string = "";
  lastNameInputValue: string = "";
  ageInputvalue: number = 0;

  constructor() { }

  ngOnInit(): void {
  }

  addRecord() {
    let person: Person = {
      'firstName': this.firstNameInputValue,
      'lastName': this.lastNameInputValue,
      'age': this.ageInputvalue
    }

    this.addPerson.emit(person);
  }

}
```

Examining the above code, we can see that we have a variable called addPerson with a decorator `@Output('addPerson')` being placed above it. The addPerson variable should be of the EventEmitter type, with a generic that specifies what type the event object that will be emitted should be. In our case, we want the event object to be a Person object.

Whenever the addRecord function is invoked (in this case, by pressing a button), it will leverage the `emit(...)` function that belongs to the EventEmitter object in order to emit an event called `addPerson`, with the event object being a Person object.

In the parent class's HTML template:

```html
<app-form (addPerson)="onAddPerson($event)" *ngIf="formComponentShouldBeDisplayed"></app-form>
```

- The important line here is the `(addPerson)="onAddPerson($event)"`
- We are binding an event called addPerson, such that whenever it occurs, it will invoke the onAddPerson function defined in our parent class
- The `$event` argument is referring to the event object itself, which happens to be a Person object