Components

So, what are they?

Components are **the heart of Angular**. They are:

- The building blocks of applications
- Self-contained

They have:

- **Logic**. It's a class with functions and state
- **Views**. Templates inline or through a URL
- **Styles**. CSS defined inline or through a URL

My first component

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

@Component({
    selector: 'hello-world',
    template: 'My first component says: {{message}}'
})
export class HelloWorldComponent {
    @Input() message: string = 'Hello world';
}
```

Use this component:

```
<hello-world></hello-world>
```

Components promote reusability

Think of reusable components such as...

- An autocompleter
- A CRUD grid table
- A tiled display
- All modal dialogs
- A simple loading indicator
- Tabbed control

A reusable loading component

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

@Component({
    selector: 'loading',
    templateUrl: 'loading.component.html'
})
export class LoadingComponent {
    @Input() message: string;
}
```

```
<div class="loading">
    <div class="icon"></div>
    <span>{{message}}</span>
</div>
```

Using the loading component

```
import { LoadingComponent } from './loading.component';
import { ListCarsComponent } from './list-cars.component';

@NgModule({
  declarations: [ /*...*/, ListCarsComponent, LoadingComponent],
  /*...*/
})
export class AppModule { }
```

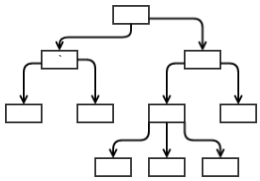
```
import { Component } from '@angular/core';
import { LoadingComponent } from './loading.component';
import { Car } from './car'; // a simple Car class with a make and type

@Component({
    selector: 'list-cars',
    templateUrl: 'list-cars.component.html',
})
export class ListCarsComponent {
    cars: Cars[] = null; /* TODO: Get list of cars from the backend*/
}
```

```
<loading message="Very busy retrieving cars" *ngIf="!cars"></loading>
<li *ngFor="let car of cars" *ngIf="cars">
    {{car.make}} {{car.type}}
</li>
```

Components can work together

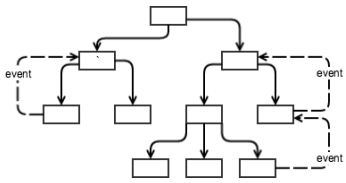
So far, we've seen that components can pass data into child components.



source: https://www.sitepoint.com/angular-2-components-inputs-outputs/

Components can work together

A component can communicate with its parent component to signal some sort of event has happened



- Create a new field of type `EventEmitter` with `@Output()` applied
- When using the component, register an event handler

source: https://www.sitepoint.com/angular-2-components-inputs-outputs/

Collaborating: Shopping cart

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

@Component({
    selector: 'shopping-cart'
    templateUrl: 'shopping-cart.component.html'
})
export class ShoppingCartComponent {
    @Output() contentChange = new EventEmitter();
    @Output() pay = new EventEmitter();

    addItemToCart() {
        this.contentChange.emit({ type: 'add', item: this.item });
    }
}
```

Using this component:

```
<shopping-cart
    (pay)="doSomthingWithPayEvent()"
    (contentChange)="doSomethingWithContent($event)"></shopping-cart>
```

Components can work together

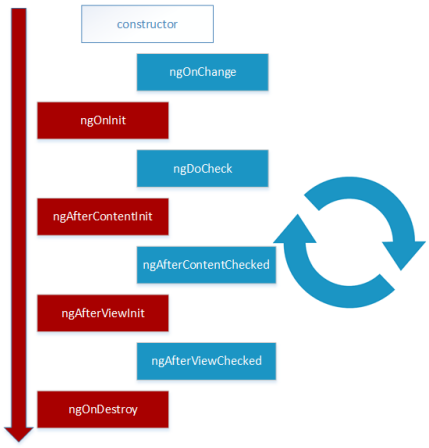
The other way around, a parent can use `@ViewChild` to access the instance of a child element.

```
@Component({ ... })
export class MainPage {
    @ViewChild(ShoppingCartComponent)
    shoppingCart: ShoppingCartComponent;

    pay() {
        this.shoppingCart.clear();
    }
}
```

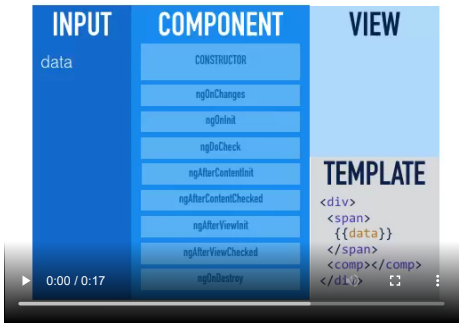
Use `@ViewChildren` to access multiple element instances.

Component lifecycle hooks



Hook	Purpose
ngOnInit	Initialize the component after Angular initializes the data-bound input properties
ngOnChanges	Respond after Angular sets a data-bound input property
ngDoCheck	Detect and act upon changes that Angular doesn't detect on its own
ngOnDestroy	Cleanup just before Angular destroys the directive/component. Unsubscribe observables and detach event handlers

Hook	Purpose
ngAfterContentInit	Respond after Angular projects external content into the component's view. (see `<ng-content>`)
ngAfterContentChecked	Respond after Angular checks the content projected into the component. (see `<ng-content>`)
ngAfterViewInit	Respond after Angular initializes the component's views and child views
ngAfterViewChecked	Respond after Angular checks the component's views and child views.



source: http://myrighttocode.org/blog/typescript/angular2/angular-2-lifecycle

Lifecycle hooks

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

@Component({
    selector: 'loading',
    templateUrl: 'loading.component.html'
})
export class LoadingComponent implements OnInit {
    @Input() message: string;

    constructor() {
        console.log('constructor message:', this.message);
    }

    ngOnInit() {
        console.log('onInit message:', this.message);
    }
}
```

```
<loading message="Very busy retrieving cars" *ngIf="!cars"></loading>
<li *ngFor="let car of cars" *ngIf="cars">
    {{car.make}} {{car.type}}
</li>
```

Cleaning up resources

Because we're dealing with a SPA, remember to clean up resources in `ngOnDestroy`. This doesn't happen automagically for you.

```
ngOnInit() {
    // Open IndexedDB connections
    // Create web workers
    // Connect web sockets
    // Set intervals/timeouts
    // Subscribe to observables
}

ngOnDestroy() {
    // Close IndexedDB connections
    // Stop web workers
    // Disconnect web sockets
    // Clear intervals/timeouts
    // Unsubscribe from observables
}
```

Recap

- Components are reusable
- Communication between components is possible using `EventEmitter` and `@Output()`
- There is a lifecycle with various hooks when working with components
  - Initialize things in `ngOnInit`
  - Clean things up in `ngOnDestroy`

Lab time!

Create two components, one for the form and one for the list. Make them communicate through their common parent, `AppComponent`.

```
<contact-form (add)="handleAdd($event)"></contact-form>

<contact-list></contact-list>
```