HOSTLISTENER & HOSTBINDING

HOSTLISTENER

We created, `ccCardHover`, which simply turned the background color gray for every element it's attached to.

But as the name of the directive implies we need a way of DETECTING if the user is hovering over the host element.

Angular makes this easy with the `@HostListener` decorator.

This is a function decorator that accepts an event name as an argument. When that event gets fired on the host element it calls the associated function.

So if we add this function to our directive class:

```
@HostListener('mouseover') onHover() {
  window.alert("hover");
}
```

Hovering over the host element would trigger an alert popup.

Lets change our directive to take advantage of the `@HostListener`.

```
import { HostListener } from '@angular/core'
.
.
.
class CardHoverDirective {
  constructor(private el: ElementRef,
              private renderer: Renderer) {

      // renderer.setElementStyle(el.nativeElement, 'backgroundColor',
      'gray');
  }

  @HostListener('mouseover') onMouseOver() {
    let part = this.el.nativeElement.querySelector('.card-text')
    this.renderer.setElementStyle(part, 'display', 'block');
  }
}
```

- We've removed the code to render the background color to gray.
- We decorate a class method with `@HostListener` configuring it to call the function on every `mouseover` events.
- We get a reference to the DOM element that holds the jokes punchline.

- We set the display to block so that element is shown.

For the above to work we need to change our `JokeComponent` template so the joke is hidden using the display style property, like so:

```
<div class="card card-block" ccCardHover>
  <h4 class="card-title">{{data.setup}}</h4>
  <p class="card-text"
     [style.display]="'none'">{{data.punchline}}</p>

</div>
```

- We change the `p` element so it's hidden via a style of `display: none`, this is so we can *show* the element by setting the style to `display: block`.
- We've also removed the button, so the only way to show the punchline is via hovering over the card.

As well as showing the punchline on a `mouseover` event we also want to *hide* the punchline on a `mouseout` event, like so:

```
class CardHoverDirective {
  constructor(private el: ElementRef,
              private renderer: Renderer) {
    // renderer.setElementStyle(el.nativeElement, 'backgroundColor', 'gray');
  }

  @HostListener('mouseover') onMouseOver() {
    let part = this.el.nativeElement.querySelector('.card-text');
    this.renderer.setElementStyle(part, 'display', 'block');
  }

  @HostListener('mouseout') onMouseOut() {
    let part = this.el.nativeElement.querySelector('.card-text');
    this.renderer.setElementStyle(part, 'display', 'none');
  }
}
```

HOSTBINDING

As well as listening to output events from the host element a directive can also bind to input properties in the host element with `@HostBinding`.

This directive can change the properties of the host element, such as the list of classes that are set on the host element as well as a number of other properties.

Using the `@HostBinding` decorator a directive can link an internal property to an input property on the host element. So if the internal property changed the input property on the host element would also change.

We first need something, a property on our directive which we can use as a source for binding.

We'll create a boolean called `ishovering` and in our `onMouseOver()` and `onMouseOut()` functions we'll set this to true and false accordingly, like so:

```
class CardHoverDirective {
  private ishovering: boolean;

  constructor(private el: ElementRef,
              private renderer: Renderer) {
  }

  @HostListener('mouseover') onMouseOver() {
    let part = this.el.nativeElement.querySelector('.card-text');
    this.renderer.setElementStyle(part, 'display', 'block');
    this.ishovering = true;
  }

  @HostListener('mouseout') onMouseOut() {
    let part = this.el.nativeElement.querySelector('.card-text');
    this.renderer.setElementStyle(part, 'display', 'none');
    this.ishovering = false;
  }
}
```

- We've created a boolean called `ishovering`.
- We set our boolean to `true` when we are being hovered over and `false` when we are not.

Now we need to *link* this source property to an input property on the host element, we do this by decorating our `ishovering` boolean with the `@HostBinding` decorator.

The `@HostBinding` decorator takes one parameter, the name of the property on the host element which we want to bind to.

If you remember we can use the alternative `ngClass` syntax by binding to the `[class.<class-name>]` property. Lets add the `card-outline-primary` class to our host element when the `ishovering` boolean is `true`.

```
import { HostBinding } from '@angular/core'
.
.
.
class CardHoverDirective {
  @HostBinding('class.card-outline-primary') private ishovering: boolean;

  constructor(private el: ElementRef,
              private renderer: Renderer) {
    // renderer.setElementStyle(el.nativeElement, 'backgroundColor',
'gray');
  }
```

```
  @HostListener('mouseover') onMouseOver() {
    let part = this.el.nativeElement.querySelector('.card-text');
    this.renderer.setElementStyle(part, 'display', 'block');
    this.ishovering = true;
  }

  @HostListener('mouseout') onMouseOut() {
    let part = this.el.nativeElement.querySelector('.card-text');
    this.renderer.setElementStyle(part, 'display', 'none');
    this.ishovering = false;
  }
}
```

SUMMARY

By using the `@HostListener` and `@HostBinding` decorators we can both listen to output events from our host element and also bind to input properties on our host element as well.