Cheat Sheet – Components & Databinding

Component Metadata

Components are normal TypeScript/ JS classes, “transformed” into components by adding **Metadata**.

```
@Component({
     selector: ‘my-selector’,
     template: `
     <h1>A heading!</h1>
     `
})
```

The @Component decorator (which applies the metadata) allows the application of a variety of different metadata configurations.
The **template** metadata is always required, since that makes up a component. Often times (always if not using routing), you’ll also need the **selector** metadata.

For a complete overview over the available metadata which you may add inside the @Component decorator, visit this link: https://angular.io/docs/ts/latest/api/core/index/ComponentMetadata-class.html

Templates & Styles

Templates and **Styles** may either be specified inline (i.e. inside the .ts file) or as separate files (.html / .css).
In the latter case, in order to use relative paths, make sure to add the `moduleId: module.id` metadata to the @Component decorator. Otherwise Angular 2 is not able to keep the relative path and find the template/ style file when running in the browser.

View Encapsulation

Angular 2, by default, emulates the Shadow DOM behavior to apply styling to components.

This means, that styles are only applied to the elements of a component, even if the style definition would meet a HTML element outside of that component.
Visit this link to learn more:
http://blog.thoughtram.io/angular/2015/06/29/shadow-dom-strategies-in-angular2.html

Databinding

Databinding allows you to communicate between the component/ class body and the template, as well as between different parts of your application.
Angular 2 knows four forms of Databinding:

1. String interpolation

```
{{property_resolving_to_string}}
```

2. Property Binding

```
<img [src]="img_src_path">
```

3. Event Binding

```
<button (click)="onClick()">
```

4. Two-Way Binding

```
<input type="text" [(ngModel)]="myModel.name">
```

Also consult the official Angular 2 Cheat Sheet, which goes into more detail about the different syntaxes etc: https://angular.io/cheatsheet

Local References

Inside templates you may create local references like this:

```
<div #myDiv>
```

This will create a reference to the DIV HTML element which you may use throughout the template (NOT inside the component class body!).
If you want to get a reference to this element in your component class body, get it by using @ViewChild, like this:

```
@ViewChild('myDiv') referenceToDivElement;
```

Component Lifecycle

Angular 2 Components follow a lifecycle when created (which is taken care of by Angular 2).

Consult this link for more information on the different hooks and when they are reached: https://angular.io/docs/ts/latest/guide/lifecycle-hooks.html