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

@Component({
  selector: 'ns-race',
  templateUrl: 'race/race.html'
})
export class RaceComponent {

  race: any;

  constructor(racesService: RacesService) {
    racesService.get()
      .then(race => this.race = race);
  }
}
```

And the template looks like this:

```
<div>
  <h2>{{ race.name }}</h2>
  <div>{{ race.status }}</div>
  <div *ngFor="let pony of race.ponies">
    <ns-pony [pony]="pony"></ns-pony>
  </div>
</div>
```

If you already know AngularJS 1.x, the template should look familiar, with the same expression in curly braces `{{ }}`, that will be evaluated and replaced by the according value. Some things have changed though: no more `ng-repeat` for example. I don't want to go too deep for now, merely just give you a feel of what the code looks like.

A component is a very isolated piece of your app. Your app *is* a component like the others.

You will group components in one or several coherent entities, called modules (Angular Modules, not ES6 Modules).

In a perfect world, you will also take available modules from the community and just put them in your app, and be able to enjoy their features.

Such modules can offer UI components, or drag and drop capability, or validation for your forms, or whatever you can think of.

In the next chapters, we are going to explore how to get started, how to build a small component, your first module and the templating syntax.

There is another concept that is at the core, and that is Dependency injection (often called by its little name, DI). It is a very powerful pattern, and you will quickly get used to it after reading the