by a fake (mock) one, to test our component.

If we use the Dependency Injection (DI) pattern, we delegate the creation of the `RaceService` to the framework, and we simply ask for an instance. The framework is now in charge of the creation of the dependency, and, well, injects it:

```
class RaceList {
  constructor(raceService) {
    this.raceService = raceService;
    this.raceService.list()
      .then(races => this.races = races);
  }
}
```

Now, when we test this class, we can easily pass a fake service to the constructor:

```
// in a test
const fakeService = {
  list: () => {
    // returns a fake promise
  }
};
const raceList = new RaceList(fakeService);
// now we are sure that the race list
// is the one we want for the test
```

But how does the framework know what to inject in the constructor? Good question! AngularJS 1.x relied on the parameter's names, but it had a severe limitation, because minification of your code would have changed the param name… You could use the array syntax to fix this, or add a metadata to the class:

```
RaceList.$inject = ['RaceService'];
```

We had to add some metadata for the framework to understand what classes needed to be injected with. And that's exactly what type annotations give: a metadata giving the framework a hint it needs to do the right injection. In Angular, using TypeScript, we can write our `RaceList` component like: