Services and Providers

- A service is nothing more than a class in Angular 2. It remains nothing more than a class until we register it with the Angular injector.
- When you bootstrap your app, Angular creates an injector on the fly that can inject services and other dependencies throughout the app.
- You can register the service or the dependencies during when bootstrapping the app or when defining a component.
- If you have a class called `MyService`, you can register it with the Injector and then you can inject it everywhere:

```
bootstrap(App, [MyService]); // second param is an array of providers
```

- Providers is a way to specify what services are available inside the component in a hierarchical fashion.
- A provider can be a class, a value or a factory.
- Providers create the instances of the things that we ask the injector to inject.
- `[SomeService];` is short for `[provide(SomeService, {useClass:SomeService})];` where the first param is the token, and the second is the definition object.
- A simple object can be passed to the Injector to create a Value Provider:

```
beforeEachProviders(() => {
  let someService = { getData: () => [] };
  // using `useValue` instead of `useClass`
  return [ provide(SomeSvc, {useValue: someService}) ];
});
```

- You can also use a factory as a provider.
- You can use a factory function that creates a properly configured Service:

```
let myServiceFactory = (dx: DepX, dy: DepY) => {
  return new MyService(dx, dy.value);
}

// provider definition object.
let myServiceDefinition = {
   useFactory: myServiceFactory,
   deps: [DepX, DepY]
};

// create provider and bootstrap
let myServiceProvider = provide(MyService, myServiceDefinition);
bootstrap(AppComponent, [myServiceProvider, DepX, DepY]);
```

- Defining object dependencies is simple. You can make a plain JavaScript object available for injection using a string-based token and the `@Inject` decorator:

```
var myObj = {};

bootstrap(AppComponent, [
  provide('coolObjToken', {useValue: myObj})
]);

// and you can inject it to a component

import {Inject} from 'angular2/core'
constructor(dx: DepX, @Inject('coolObjToken') config)
```

Simple Service

In this section we are going to make a simple service and use it in our root component.

Project Files

The project files for this section are in angular2-intro/project-files/angularexamples/services/simple-service;

Getting Started

Let's get started by creating a class, called `StudentSvc` that represents our service:

```
class StudentSvc {
  private students: any[];
  constructor() {
    this.students = [
      {name: 'Tom', id: 1},
      {name: 'John', id: 2},
      {name: 'Kim', id: 3},
      {name: 'Liz', id: 4}
    ];
  }
  getAll() {
    return this.students;
  }
}
```

There is nothing special about this class. It's just a class the has a method to return the list of all students. Now, we are going to make it special by decorating it with the `Injectable` decorator. But, first we need to import `Injectable` from Angular:

```
import {Injectable} from 'angular2/core';
```

After importing the `Injectable` metadata class, we can decorate our class:

```
/**
 * Student service
 */
@Injectable() // <- decorating with `Injectable`
class StudentSvc {
  private students: any[];
  constructor() {
    // ...
  }
  // ...
}
```

Now we have an injectable class and the injector would know how to create an instance of it when we need to inject it. And that's what we are going to do next. We are going to add `StudentSvc` in the list of `viewProviders` of the root component:

```
@Component({
  selector: 'app',
  templateUrl : 'templates/app.tpl.html',
  viewProviders: [StudentSvc] // <- registering the service
})
```

The last thing we need to do is to inject the service in the constructor of our root component:

```
1 class Root  {
2   private name: string;
3   private students: any[];
4   constructor (studentSvc: StudentSvc) { // <- injecting the service
5     this.name = 'Simple Service Demo';
6     this.students = studentSvc.getAll(); // <- calling the `getAll` method
7   }
8 }
```

- In the constructor, we are defining a variable to be of type `StudentSvc`. By doing that the injector will create an instance from the `StudentSvc` to be used

- And on line 6 we are calling the `getAll` method from the service to get a list of all students.

Finally, we can verify that the `getAll` method is actually called by printing the students in the template:

`app.tpl.html`

```html
<h1>{{ name }}</h1>

<ul>
  <li *ngFor="#student of students">Name: {{ student.name }}, id: {{ student.
</ul>
```

and it would output:

```
Name: Tom, id: 1
Name: John, id: 2
Name: Kim, id: 3
Name: Liz, id: 4
```