Services, Dependency Injection, and Component Lifecycle Hooks

John Papa
PRINCIPAL ARCHITECT

@john_papa www.johnpapa.net

Overview

Services

Dependency Injection

Component Lifecycle Hooks

Services

Services

A Service provides anything our application needs. It often shares data or functions between other Angular features

Services

Angular 1

- Factories
- Services
- Providers
- Constants
- Values

Angular 2

- Class

vehicle.service.ts

```
@Injectable()
export class VehicleService {
  getVehicles() {
    return [
      new Vehicle(10, 'Millenium Falcon'),
      new Vehicle(12, 'X-Wing Fighter'),
      new Vehicle(14, 'TIE Fighter')
    ];
  }
}
```

A Service is just a class

Service

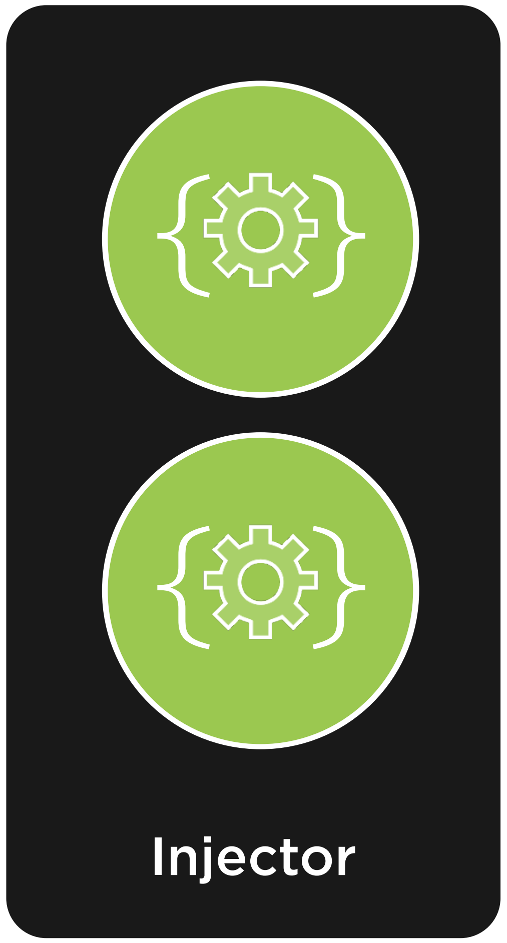
Provides something of value

Shared data or logic

e.g. Data, logger, exception handler, or message service

Demo

Services



Dependency Injection

Dependency Injection

Dependency Injection is how we provide an instance of a class to another Angular feature vehicle.component.ts

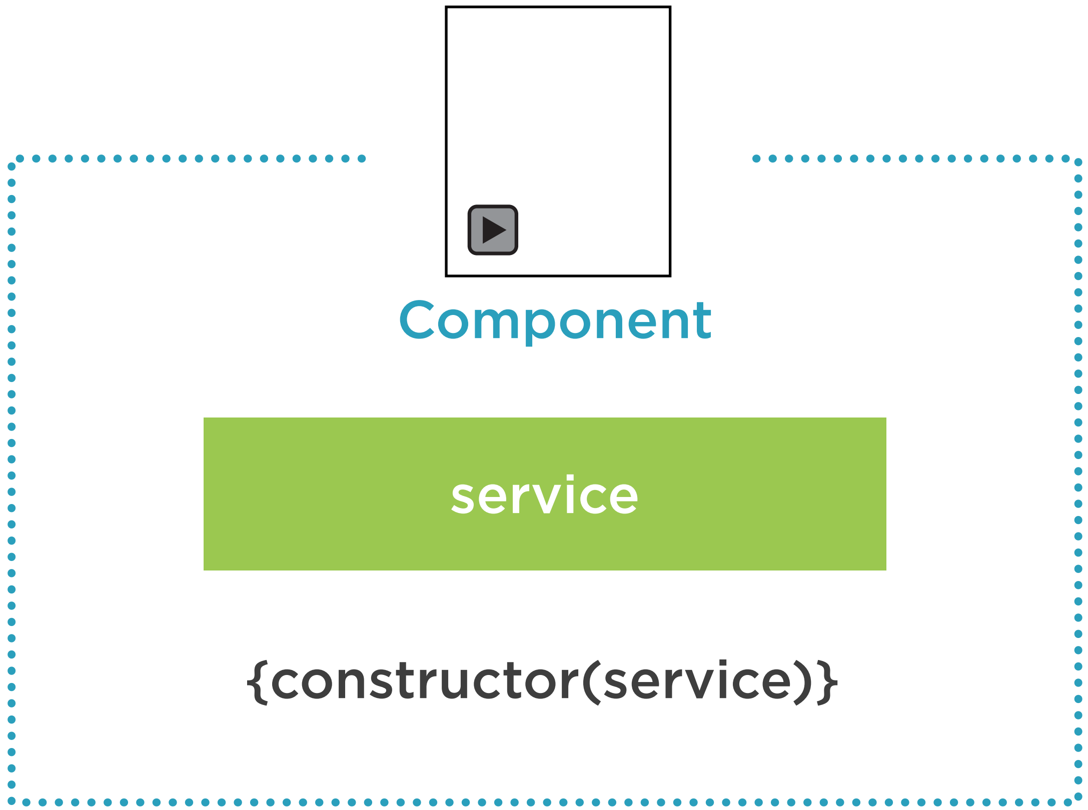
```
export class VehicleListComponent {
  vehicles: Vehicle[];

  constructor(private _vehicleService: VehicleService) {
    this._vehicleService.getVehicles()
      .subscribe(vehicles => this.vehicles = vehicles);
  }
}
```

Injecting VehicleService

Injecting a Service into a Component

Locates the service in the Angular injector

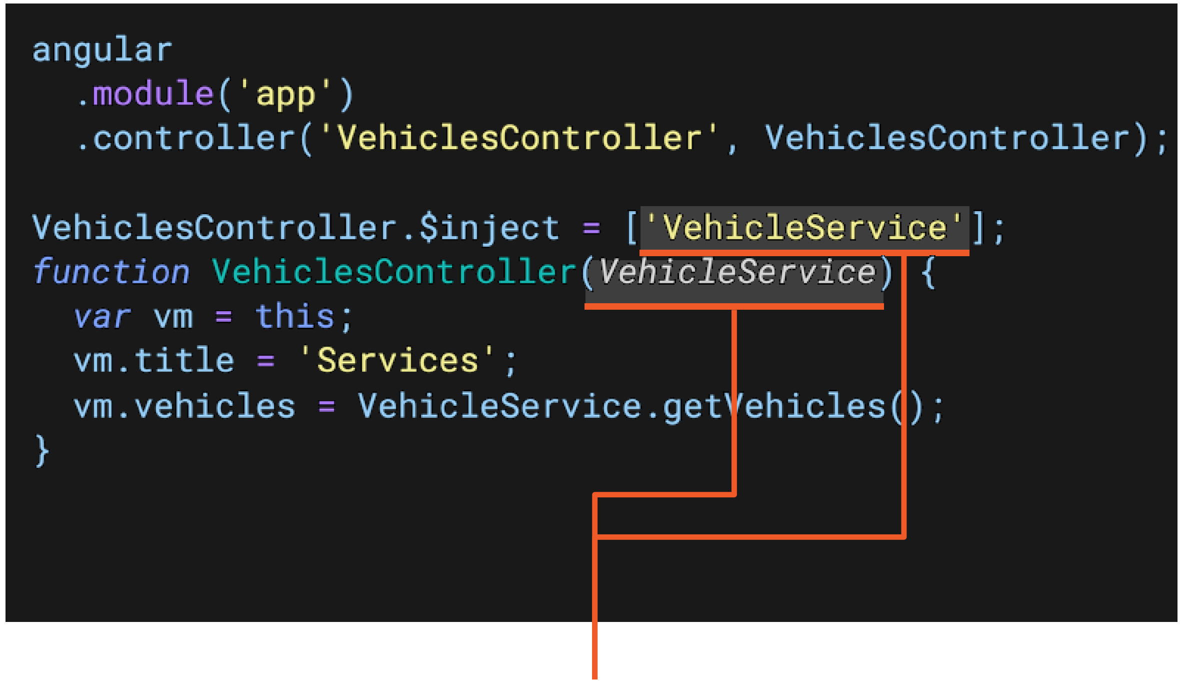
Injects into the constructor



Service is injected into the Component's constructor

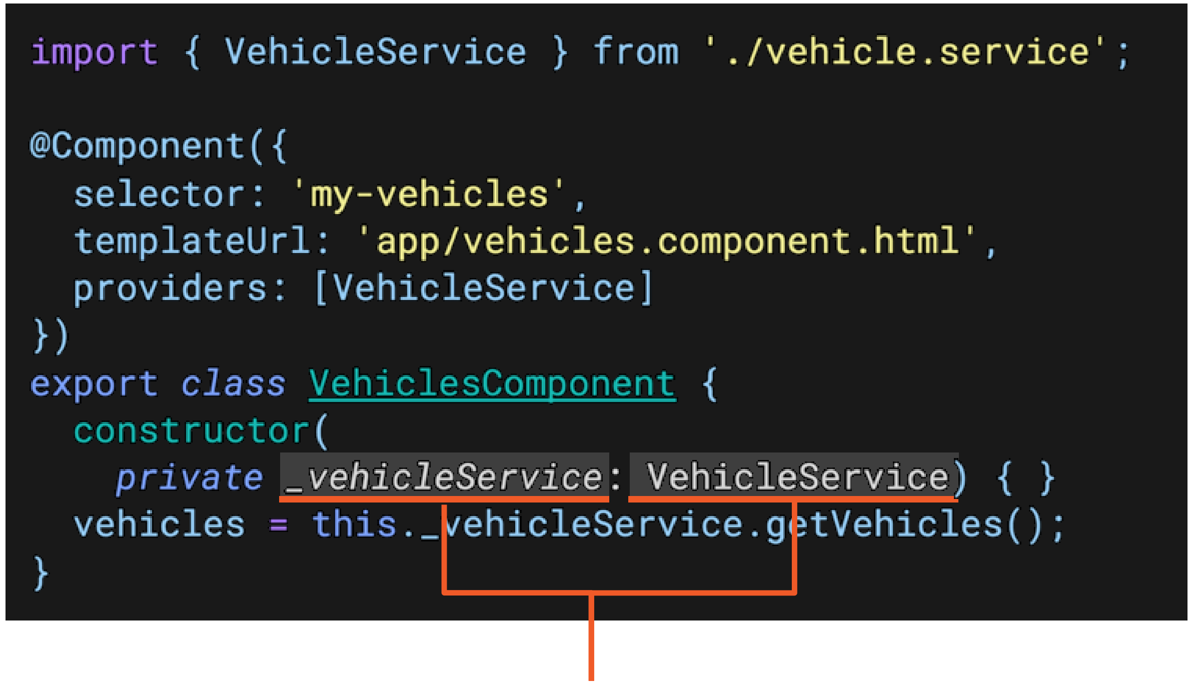
Dependency Injection Then and Now

Angular 1



```
angular
  .module('app')
  .controller('VehiclesController', VehiclesController);

VehiclesController.$inject = ['VehicleService'];
function VehiclesController(VehicleService) {
  var vm = this;
  vm.title = 'Services';
  vm.vehicles = VehicleService.getVehicles();
}
```

Angular 2



```
import { VehicleService } from './vehicle.service';

@Component({
  selector: 'my-vehicles',
  templateUrl: 'app/vehicles.component.html',
  providers: [VehicleService]
})
export class VehiclesComponent {
  constructor(
    private _vehicleService: VehicleService) { }
  vehicles = this._vehicleService.getVehicles();
}
```

vehicle.service.ts

```
@Injectable()
export class VehicleService {
  constructor(private _http: Http) { }

  getVehicles() {
    return this._http.get(vehiclesUrl)
      .map((response: Response) => <Vehicle[]>response.json().data);
  }
}
```

Provides metadata about the Injectables

Injecting http

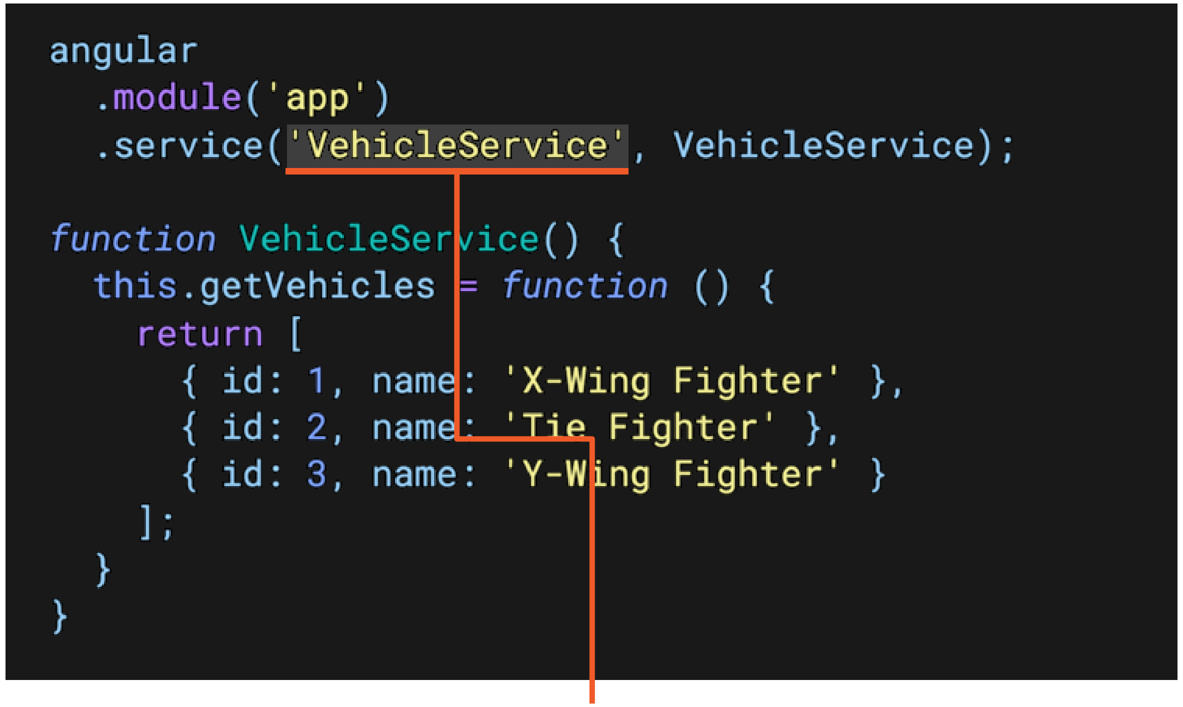
Injecting a Service into a Service

Same concept as injecting into a Component

@Injectable() is similar to Angular 1's $inject

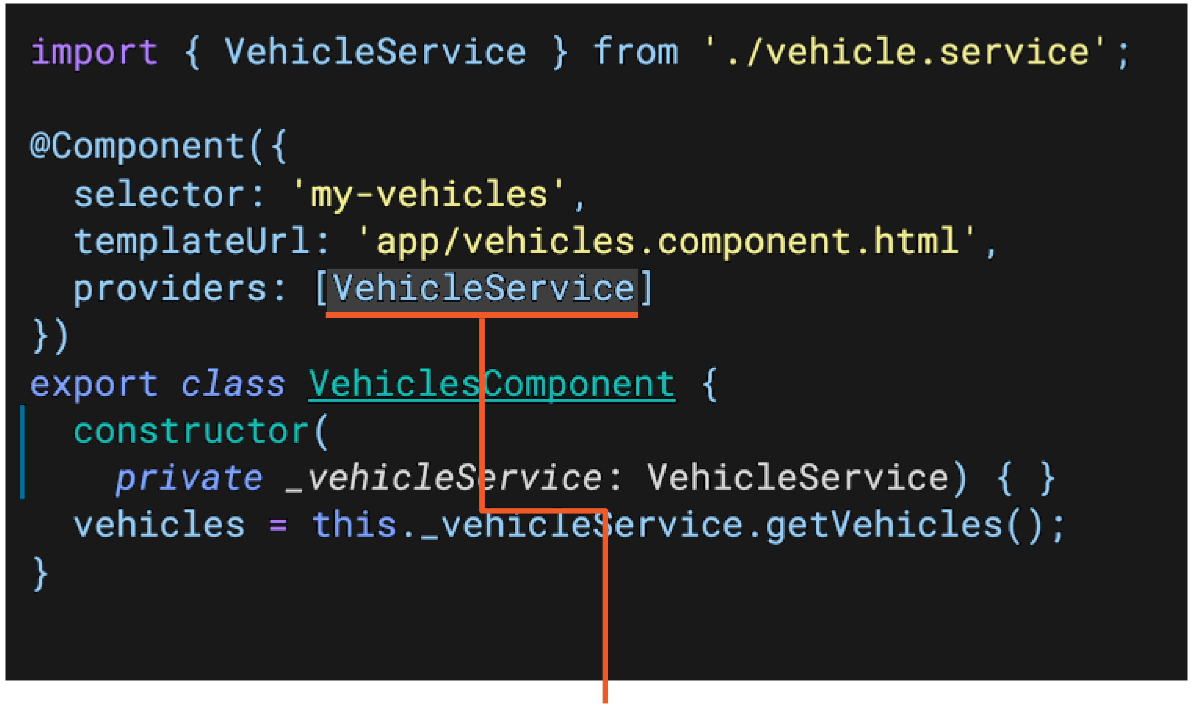
Registering Services with the Injector

Angular 1



```
angular
  .module('app')
  .service('VehicleService', VehicleService);

function VehicleService() {
  this.getVehicles = function () {
    return [
      { id: 1, name: 'X-Wing Fighter' },
      { id: 2, name: 'Tie Fighter' },
      { id: 3, name: 'Y-Wing Fighter' }
    ];
  }
}
```

Angular 2



```
import { VehicleService } from './vehicle.service';

@Component({
  selector: 'my-vehicles',
  templateUrl: 'app/vehicles.component.html',
  providers: [VehicleService]
})
export class VehiclesComponent {
  constructor(
    private _vehicleService: VehicleService) { }
  vehicles = this._vehicleService.getVehicles();
}
```

Providers

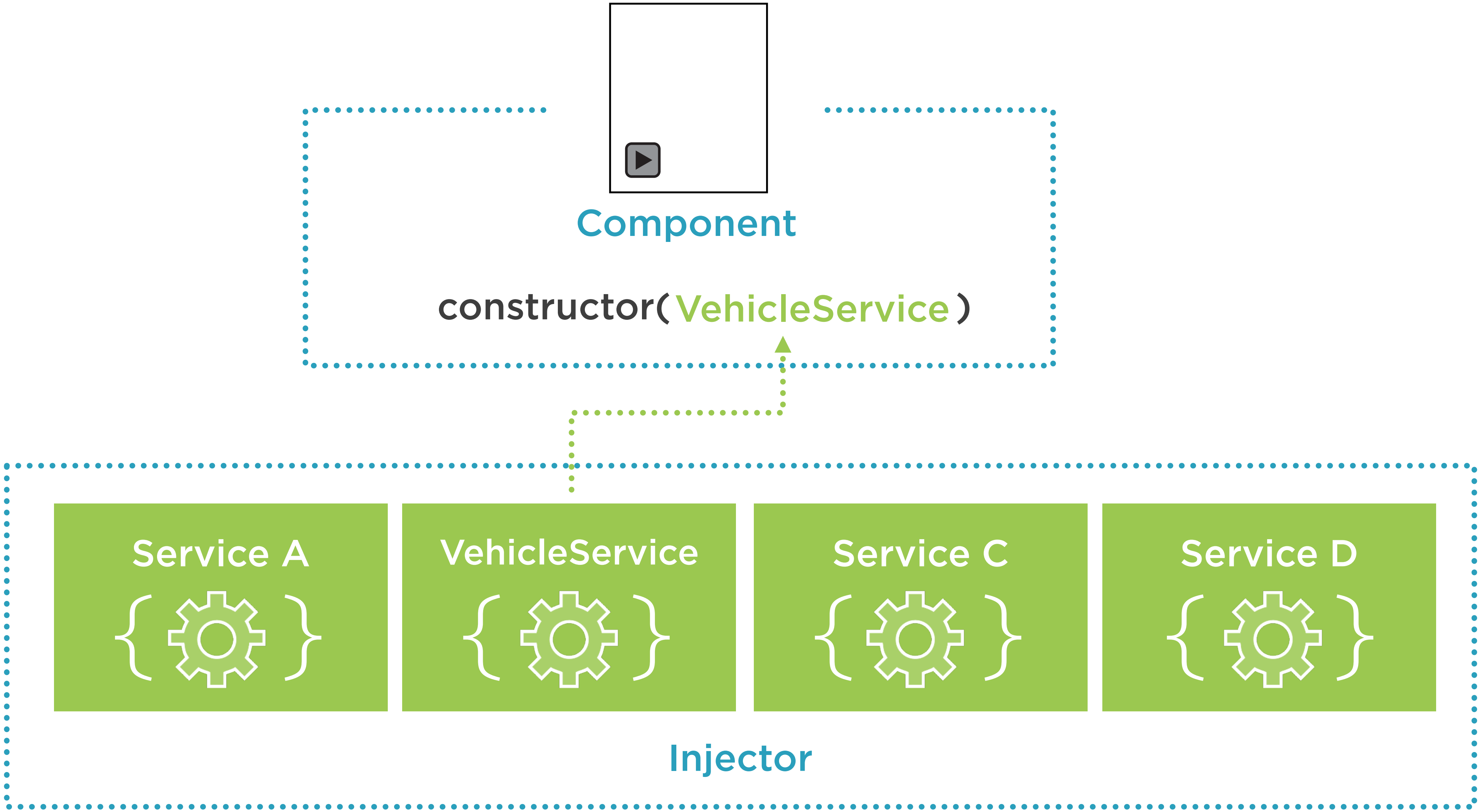
Register these Services with Angular's injector

Injection

Inject a Service into another object

```
@Component({
  selector: 'story-characters',
  templateUrl: './app/characters.component.html',
  styleUrls: ['./app/characters.component.css'],
  directives: [CharacterDetailComponent],
  providers: [HTTP_PROVIDERS, CharacterService]
})
export class CharactersComponent implements OnInit {
  @Output() changed = new EventEmitter<Character>();
  @Input() storyId: number;
  characters: Observable<Character[]>;
  selectedCharacter: Character;

  constructor(private _characterService: CharacterService) { }

  ngOnInit() {
    this.characters = this._characterService
      .getCharacters(this.storyId);
  }

  select(selectedCharacter: Character) {
    this.selectedCharacter = selectedCharacter;
    this.changed.emit(selectedCharacter);
  }
}
```

Register the service with the injector at the parent that contains all components that require the service



Component
constructor(VehicleService)
Service A
VehicleService
Service C
Service D
Injector

Component Lifecycle Hooks

Component Lifecycle Hooks

Lifecycle Hooks allow us to tap into specific moments in the application lifecycle to perform logic.

Interface

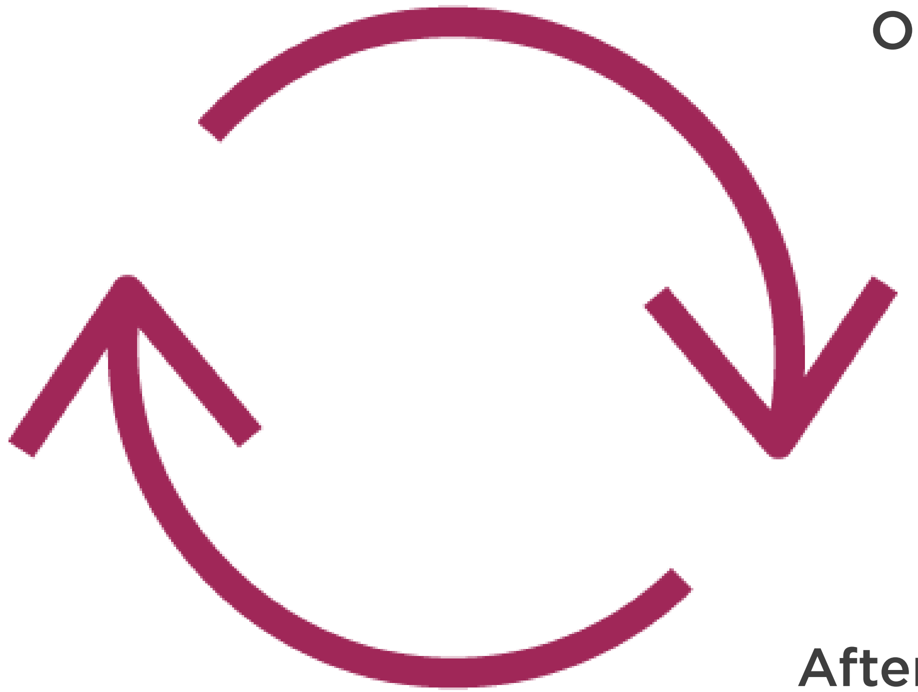
Implement the lifecycle hook's interface

Lifecycle Hooks

When the Component initializes, the ngOnInit function is executed

```
@Component({
  selector: 'story-characters',
  templateUrl: './app/characters.component.html',
  styleUrls: ['./app/characters.component.css'],
  directives: [CharacterDetailComponent],
  providers: [HTTP_PROVIDERS, CharacterService]
})
export class CharactersComponent implements OnInit {
  @Output() changed = new EventEmitter<Character>();
  @Input() storyId: number;
  characters: Observable<Character[]>;
  selectedCharacter: Character;

  constructor(private _characterService: CharacterService) { }

  ngOnInit() {
    this.characters = this._characterService
      .getCharacters(this.storyId);
  }

  select(selectedCharacter: Character) {
    this.selectedCharacter = selectedCharacter;
    this.changed.emit(selectedCharacter);
  }
}
```

Component Lifecycle Hooks



OnChanges

OnInit

AfterViewInit

OnChanges

OnDestroy

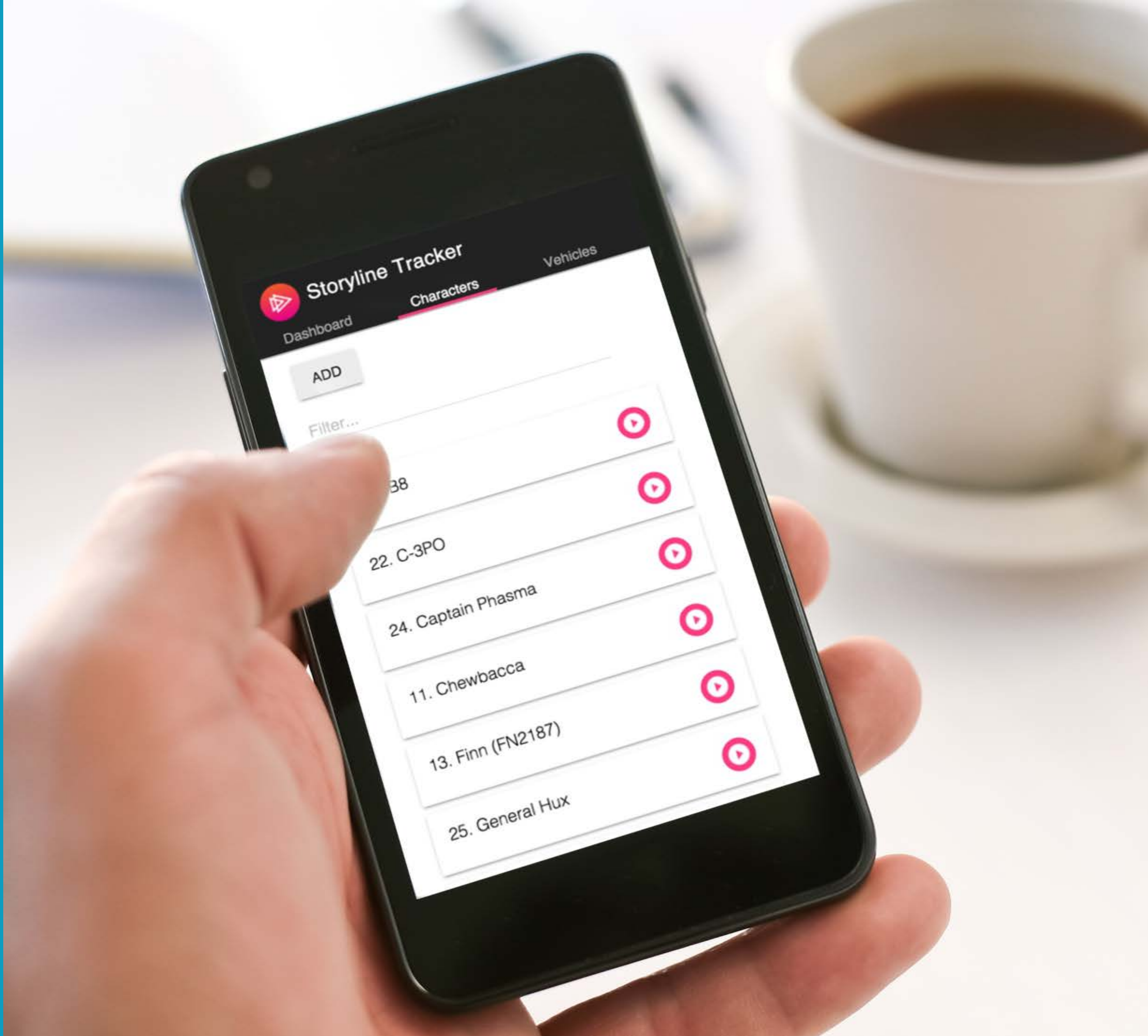
The Lifecycle Interface helps enforce the valid use of a hook

Demo

Component Lifecycle Hooks

Demo

Putting It All Together



Services, DI, and LifeCycle Hooks

Separation with Services

Sharing Instances

Registering with the Injector

Constructor Injection

Tapping into the Component's LifeCycle