ANGULAR 10

Information
Technology
Institute

Presented by
Eng./Abanoub Nabil
Teaching assistant at ITI

Quick Quiz (20 min)

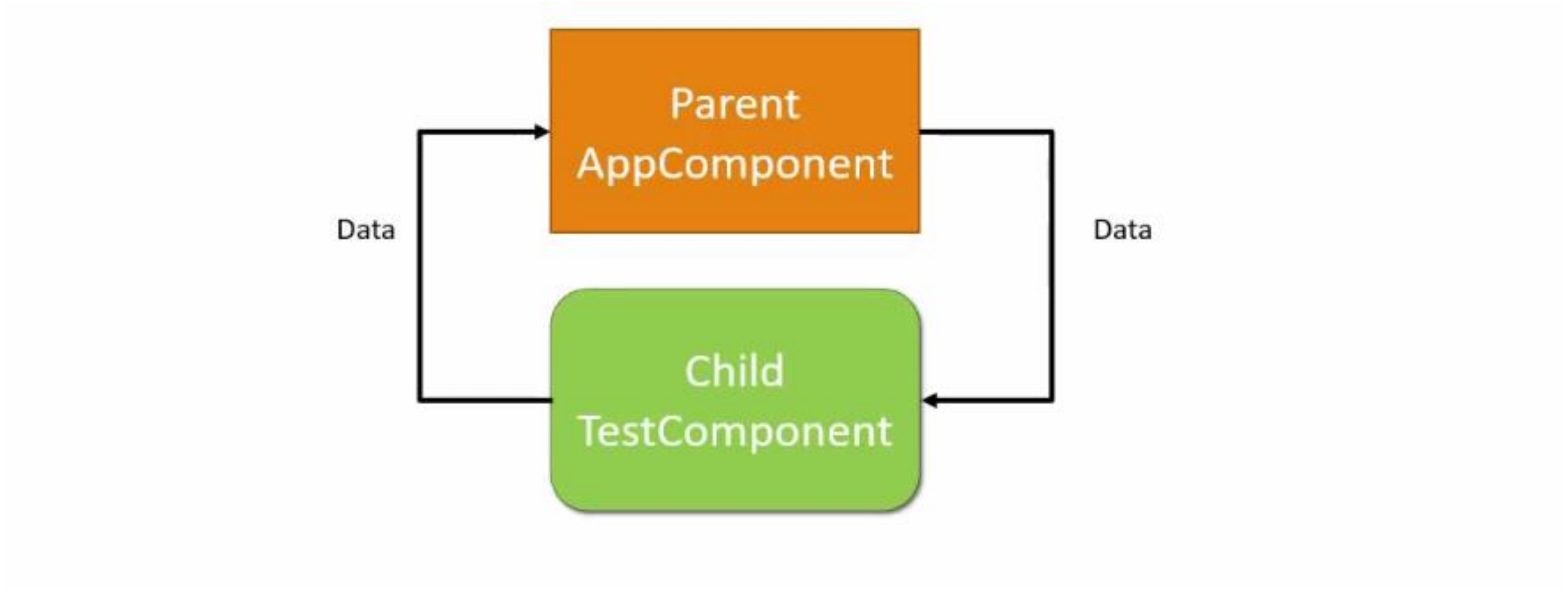
- Make a component called **DataComponent** and make a text input field inside its template.
- Make button with the value click me.

  When the text written inside the textbox is equal to true and a button clicked a div with green color appears contain the text **"hello from the true Div ☺"** and when the text is equal to false a and the button clicked a div appears with red color with text **"hello from false div ☹".**

Agenda

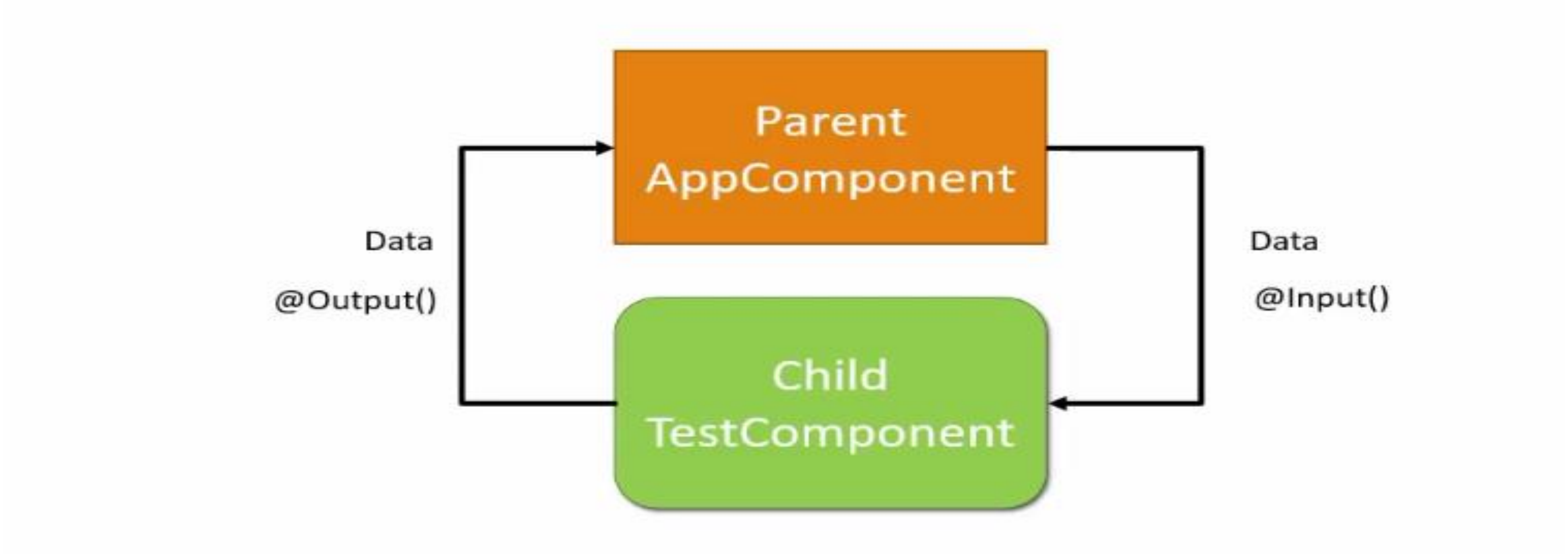
- Component interaction
- DI
- Services

Component interaction



Component interaction(cont.)

- HOW COMPONENTS INTERACT??



- The answer is using **input** and **output** decorators.

Component interaction(cont.)

- Using **input** decorator the child can accept input from the parent.
- Using **output** decorator the child will send out events to the parent to indicate something.

Component interaction(cont.)

Sending data from parent component to child component

- The parent contain the selector of the child so we will send data in the selector.
- In the **parent.ts** we will define property

```
public name="Hello Abanoub From parent";
```

- Then we will send it in the selector.

```
<app-child-component [parentData]="name">
</app-child-component>
```

- **Note** that parentData is property in the child component

Component interaction(cont.)

- Now we send data to the child component lets receive it.
- First we must declare the same property that we send data to it which is called **parentData**.
- Then we need that this is not normal property this is **input** property and we will receive its value from the parent.

- Don't forget to include

```
import {Input} from '@angular/core';
```

Component interaction(cont.)

- To define the input property

```
@Input() public parentData;
```

- Then bind the property using interpolation

```
{{parentData}}
```

- If you want to change the receiving property name

```
@Input('parentData')public name;
```

Component interaction(cont.)

Sending data from child to parent.

- In the parent component you can see that it includes child selector but **not the same** in the child component so you cannot send data the same way.

- The way you can send data from child to parent is using **events.**

Component interaction(cont.)

- **First** we should create event that we will send to the parent component.
- Create new instance of **eventEmitter** class.
- Make sure that you include **eventEmitter** class from **@angular/core**
- To send it to parent we should use **output** decorator and make sure you import it from **@angular/core**

```
@Output()
public childEvent=new EventEmitter();
```

Component interaction(cont.)

- **Second** we should fire this event on a button click.

```
<button (click)="FireEvent()">Click to send
data to parent</button>
```

- Make the body of FireEvent function

```
FireEvent()
{
this.childEvent.emit("Hello Abanoub from child");
}
```

Component interaction(cont.)

- Now we should capture the event in the parent component.

```
<app-child-component
(childEvent)="messagefromChild=$event"
app-child-component>
```

Note that messagefromChild is property in the parent component.

Component interaction(cont.)

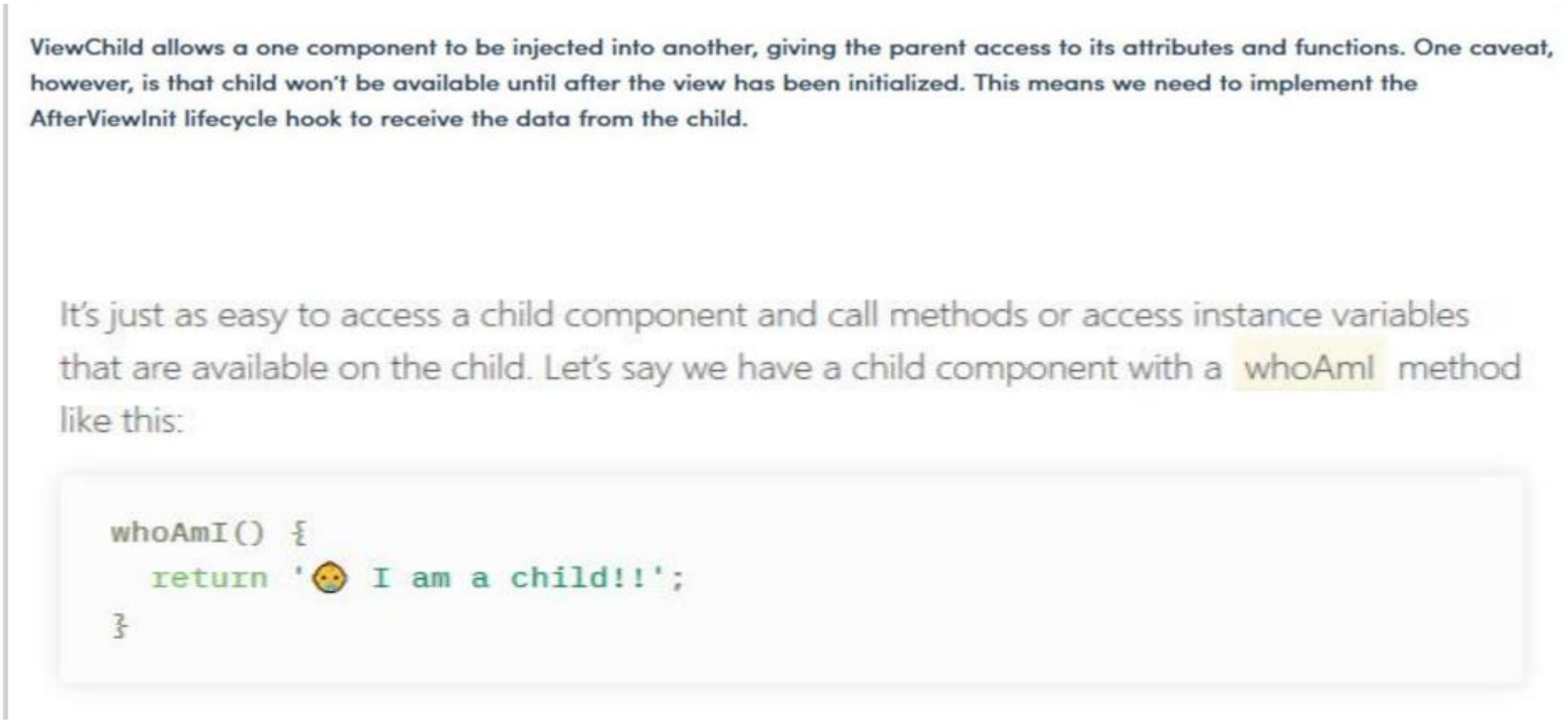
- In the parent

```
messagefromChild:string;

{{messagefromChild}}
```

Component interaction(cont.)

- Using @viewchild() to access child component:



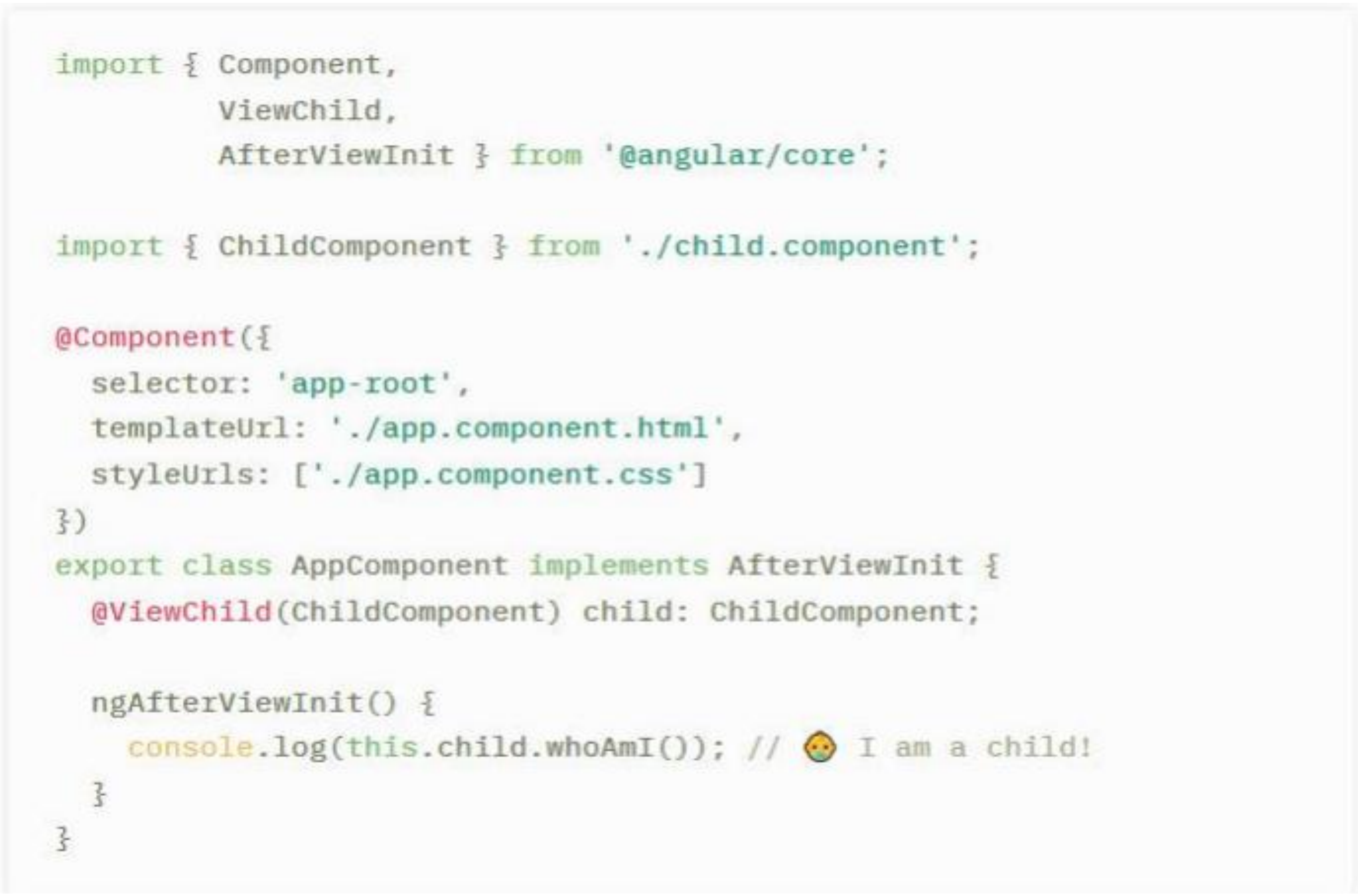
ViewChild allows a one component to be injected into another, giving the parent access to its attributes and functions. One caveat, however, is that child won't be available until after the view has been initialized. This means we need to implement the AfterViewInit lifecycle hook to receive the data from the child.

It's just as easy to access a child component and call methods or access instance variables that are available on the child. Let's say we have a child component with a `whoAmI` method like this:

```
whoAmI() {
  return '👶 I am a child!!';
}
```

Component interaction(cont.)

- l



```
import { Component,
         ViewChild,
         AfterViewInit } from '@angular/core';

import { ChildComponent } from './child.component';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements AfterViewInit {
  @ViewChild(ChildComponent) child: ChildComponent;

  ngAfterViewInit() {
    console.log(this.child.whoAmI()); // 👶 I am a child!
  }
}
```

Services

- **Principles**

  1-Don’t repeat yourself (DRY).

  2-Single responsibility principle.(**S**OLID)

Services(cont.)

- Is a class with a specific purpose.

- **Why we need it?**

  1-Share data a cross multiple components.

  2-Implement application logic.

  3-External interaction such as connecting to database.

naming convention **service.ts**

Services(cont.)

- How we using Services?????

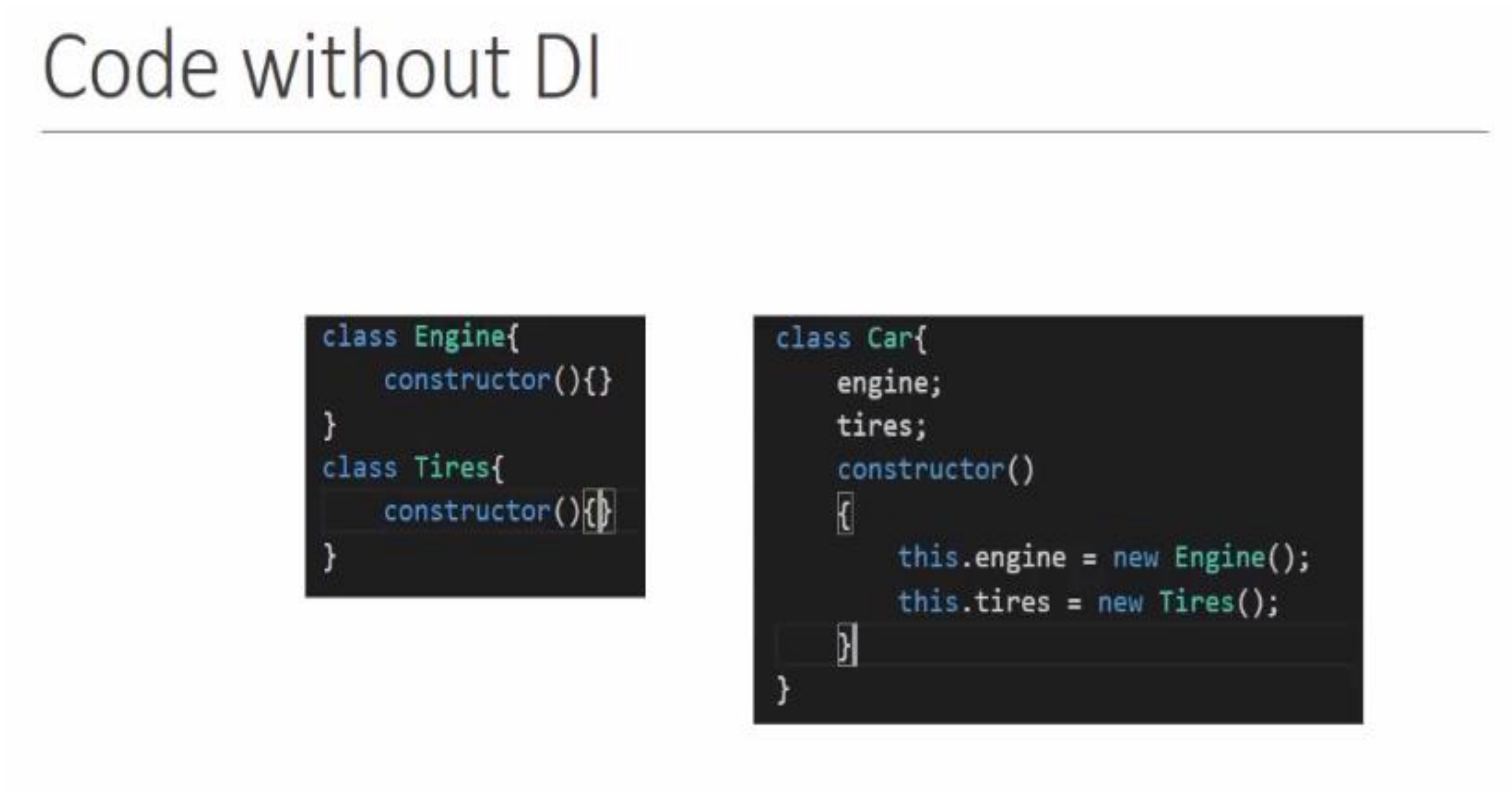
- The answer is by using

Dependency injection

Services(cont.)

- Code without DI –Drawbacks.
- DI as a design pattern.
- DI as a framework as angular provide.

Services(cont.)



Code without DI

```
class Engine{
    constructor(){}
}
class Tires{
    constructor(){}
}
```

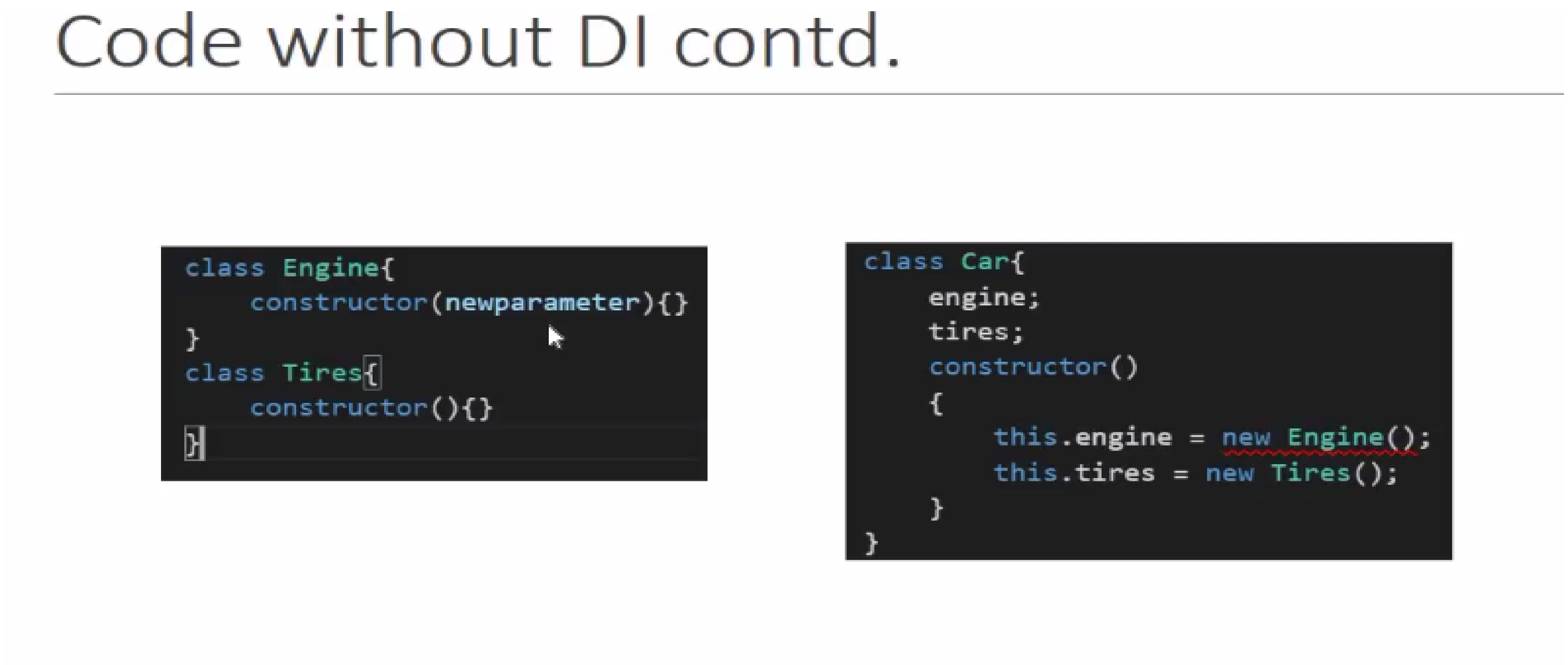
```
class Car{
    engine;
    tires;
    constructor()
    {
        this.engine = new Engine();
        this.tires = new Tires();
    }
}
```

Services(cont.)

- **Example explanation.**
- We have three classes (Car , Engine , Tiers).
- Assume that we need engine and four tiers to build a car so car has two dependencies engine and a tier.
- In the car class in constructor itself it creates instances of engine and Tier so when we create car object an object of both engine and tier will be created too.

Services(cont.)

- But there is a problem with the previous code.
- Lets assume that engine constructor accept an parameter such as engine type petrol of diesel.



Code without DI contd.

```
class Engine{
     constructor(newparameter){}
}
class Tires{
     constructor(){}
}
```

```
class Car{
     engine;
     tires;
     constructor()
     {
          this.engine = new Engine();
          this.tires = new Tires();
     }
}
```

Services(cont.)

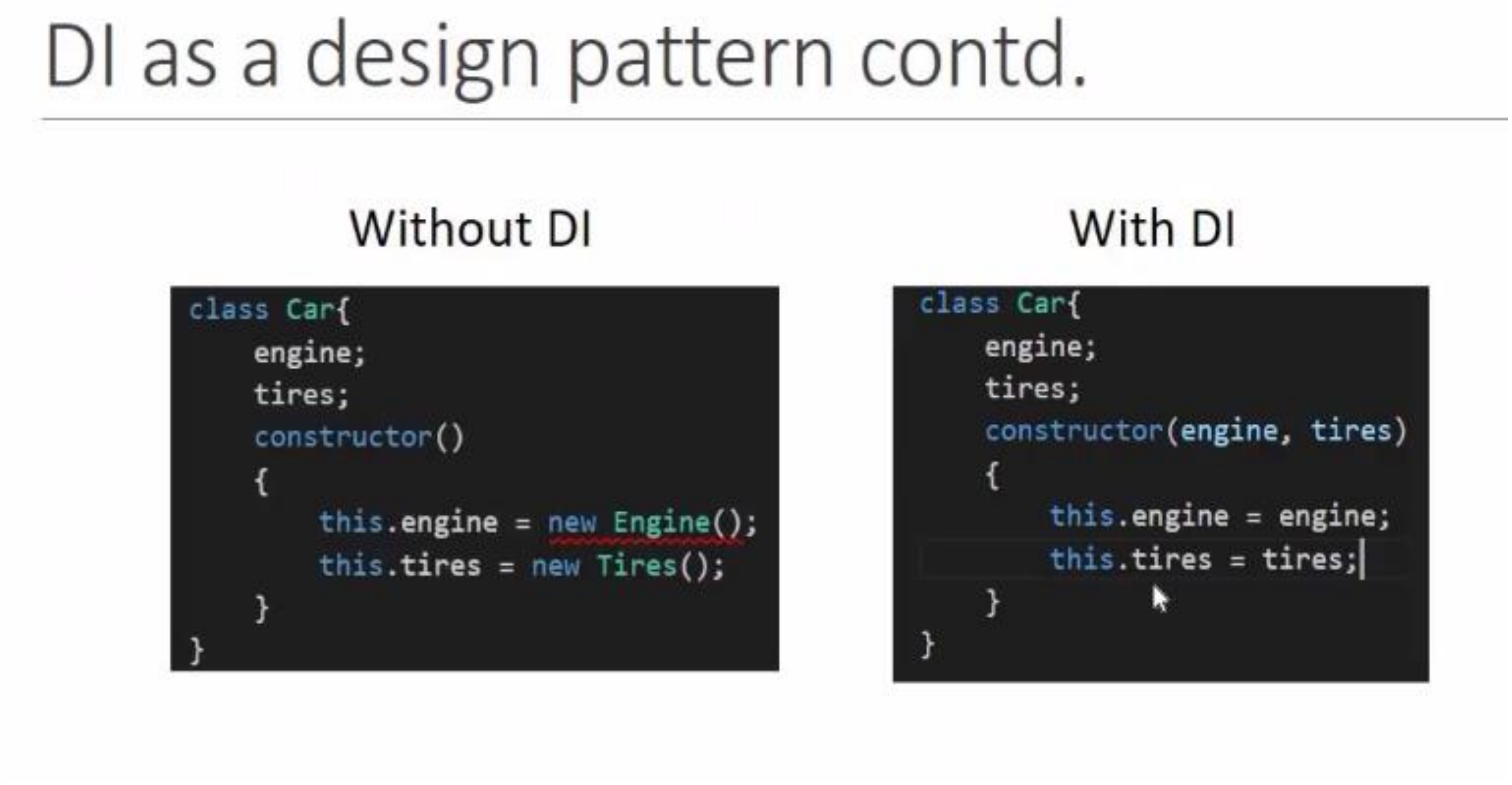
- When we changed the engine class the car class broken to repair it we need to pass parameter to engine constructor.
- So the first drawback is that the code is not flexible because any time the dependences change the car class should be change also.
- The second drawback is that this code is not suitable for testing.

Services(cont.)

DI as a design pattern.

- DI is a coding pattern which a class receive its dependencies from external source rather than creating it itself.

Services(cont.)



DI as a design pattern contd.

Without DI

```
class Car{
    engine;
    tires;
    constructor()
    {
        this.engine = new Engine();
        this.tires = new Tires();
    }
}
```

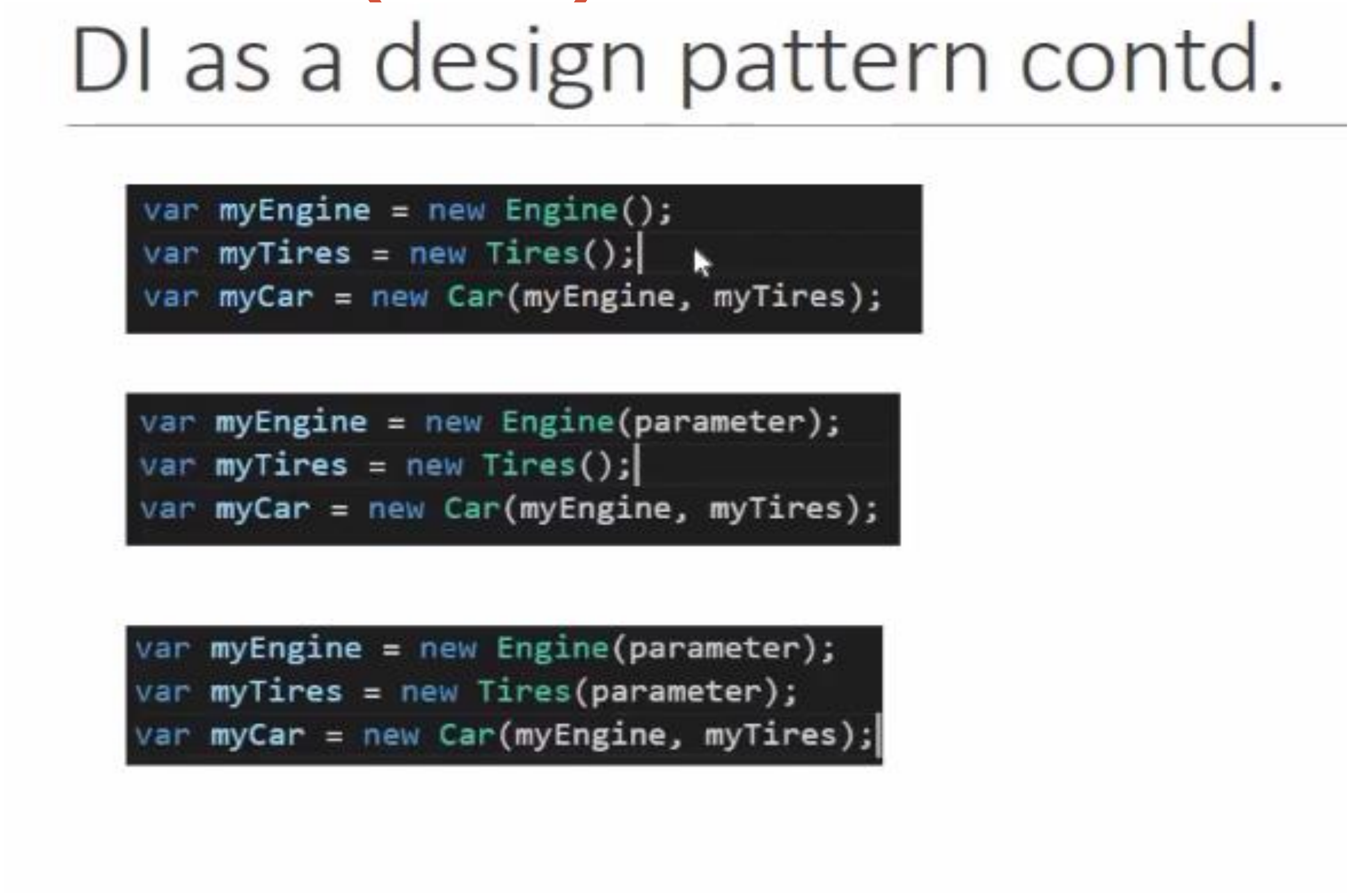
With DI

```
class Car{
    engine;
    tires;
    constructor(engine, tires)
    {
        this.engine = engine;
        this.tires = tires;
    }
}
```

Services(cont.)

- If you see the previous example we removed the dependencies from inside the constructor to the parameters of the constructor

Services(cont.)



DI as a design pattern contd.

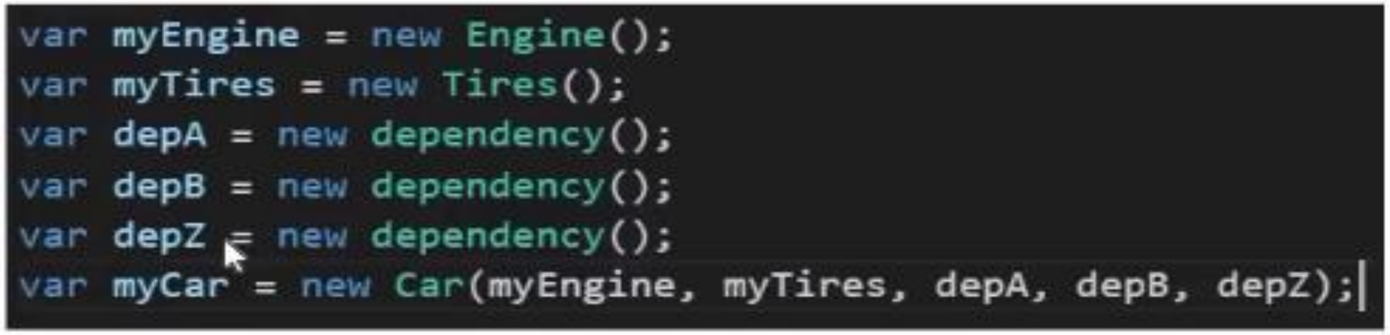
```
var myEngine = new Engine();
var myTires = new Tires();
var myCar = new Car(myEngine, myTires);
```

```
var myEngine = new Engine(parameter);
var myTires = new Tires();
var myCar = new Car(myEngine, myTires);
```

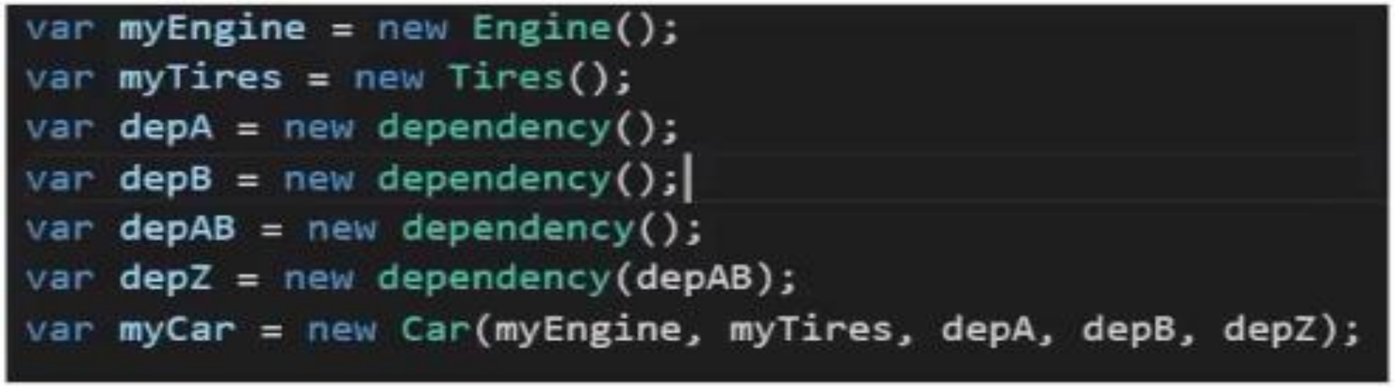
```
var myEngine = new Engine(parameter);
var myTires = new Tires(parameter);
var myCar = new Car(myEngine, myTires);
```

Services(cont.)

- But what if car has ten or more dependencies we should create all of them first then pass them as a parameter.



```
var myEngine = new Engine();
var myTires = new Tires();
var depA = new dependency();
var depB = new dependency();
var depZ = new dependency();
var myCar = new Car(myEngine, myTires, depA, depB, depZ);
```



```
var myEngine = new Engine();
var myTires = new Tires();
var depA = new dependency();
var depB = new dependency();
var depAB = new dependency();
var depZ = new dependency(depAB);
var myCar = new Car(myEngine, myTires, depA, depB, depZ);
```

Services(cont.)

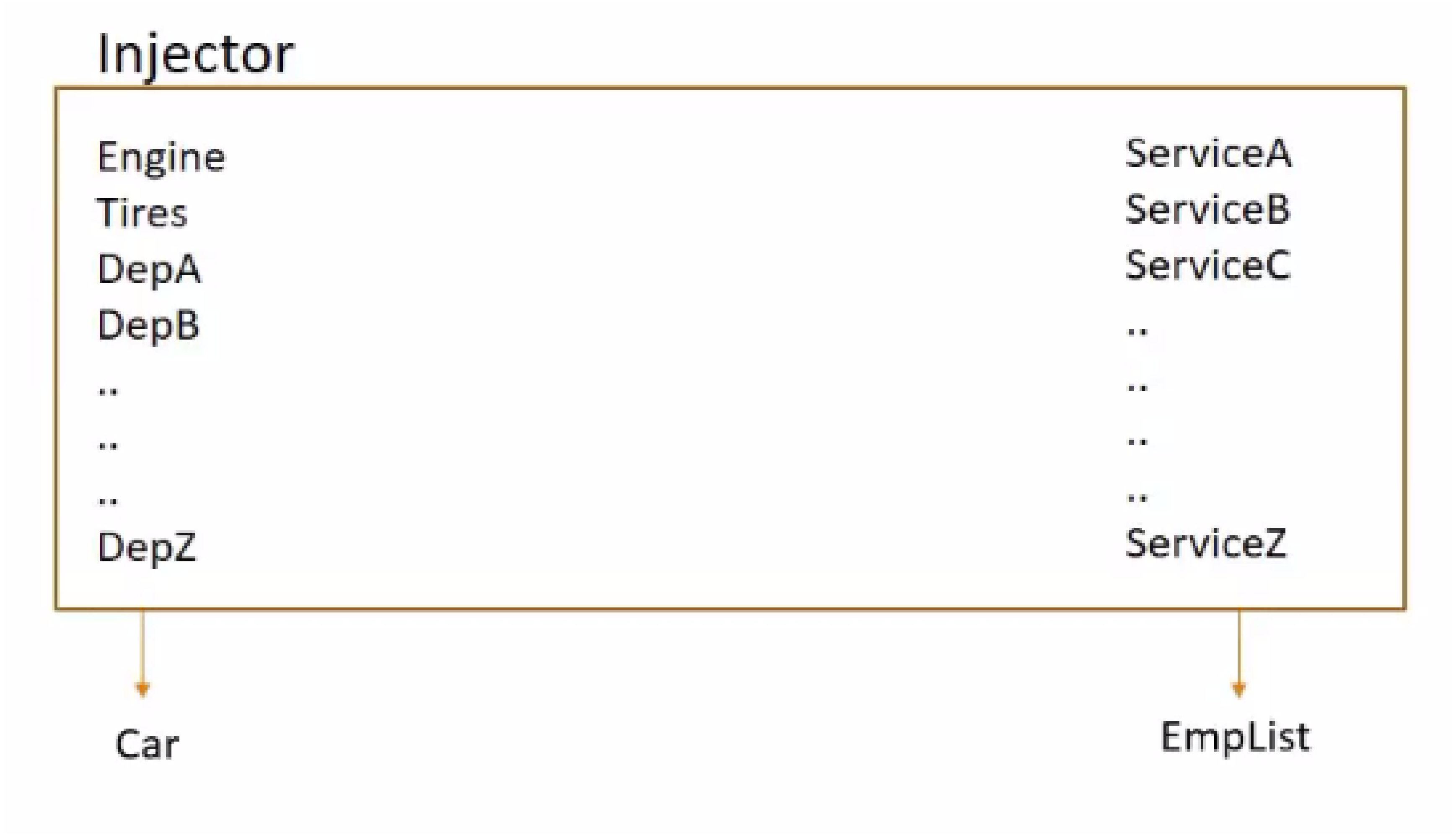
- So with the previous way it will be very difficult for a developer.
- This is where angular DI framework try to solve it.

Services(cont.)

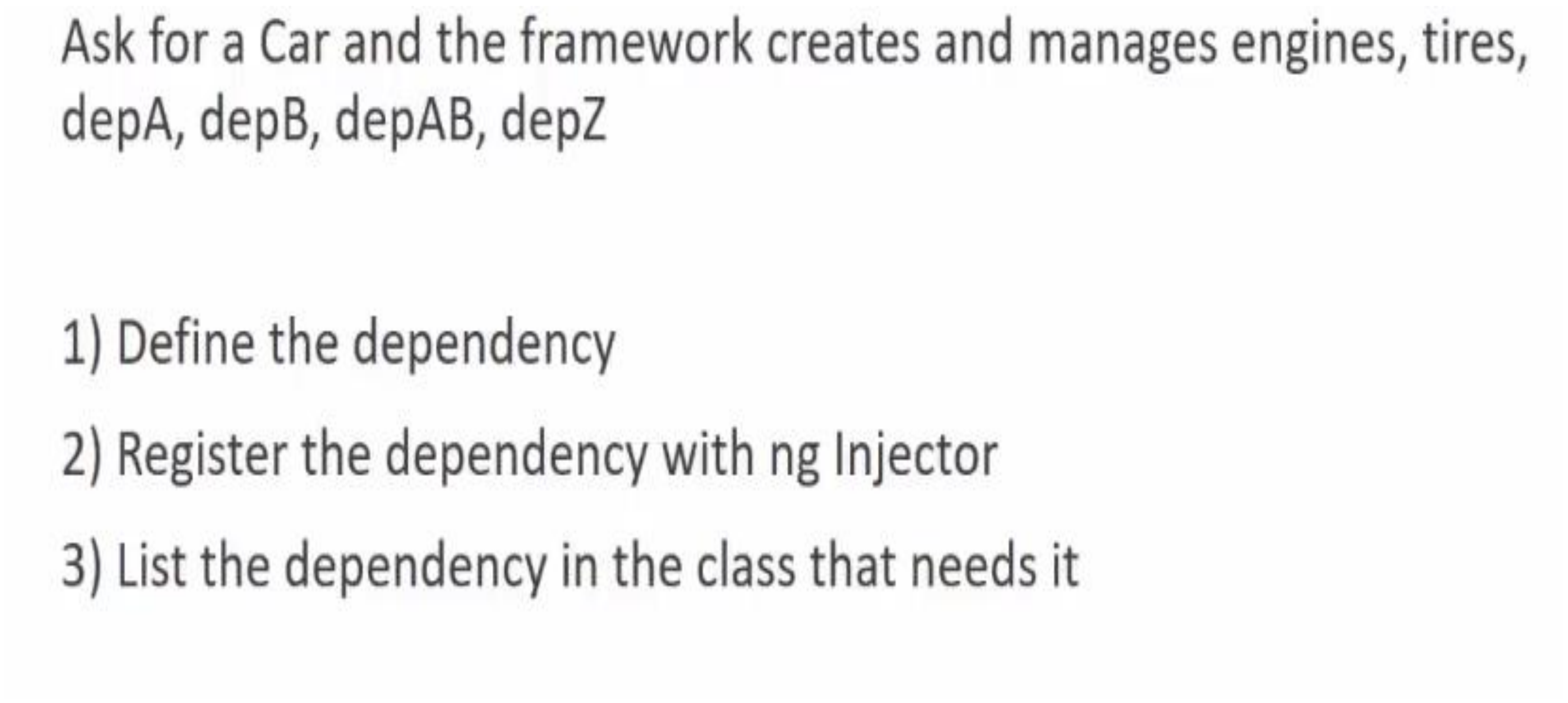
DI as a framework

- DI framework is such as something like injector that you will register all dependencies so the **injector like a container** of the dependencies so if you want car for example ask the injector for it and he will provide it for you.

Services(cont.)



Services(cont.)



Ask for a Car and the framework creates and manages engines, tires, depA, depB, depAB, depZ

1) Define the dependency

2) Register the dependency with ng Injector

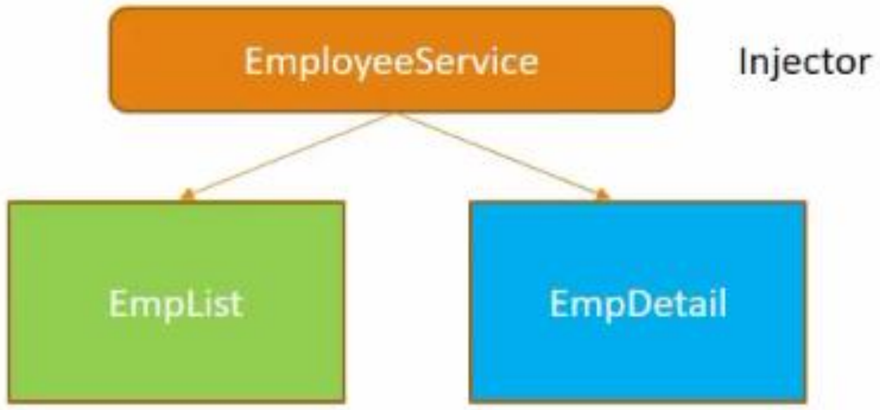
3) List the dependency in the class that needs it

Services(cont.)

1) Define the EmployeeService class

2) Register with Injector

3) Declare as dependency in EmpList and EmpDetail



Services(cont.)

- **Implement the three steps.**

1-Define the service

we can define the service or declare it using angular CLI

Run the command:

```
ng g s ServiceName
```

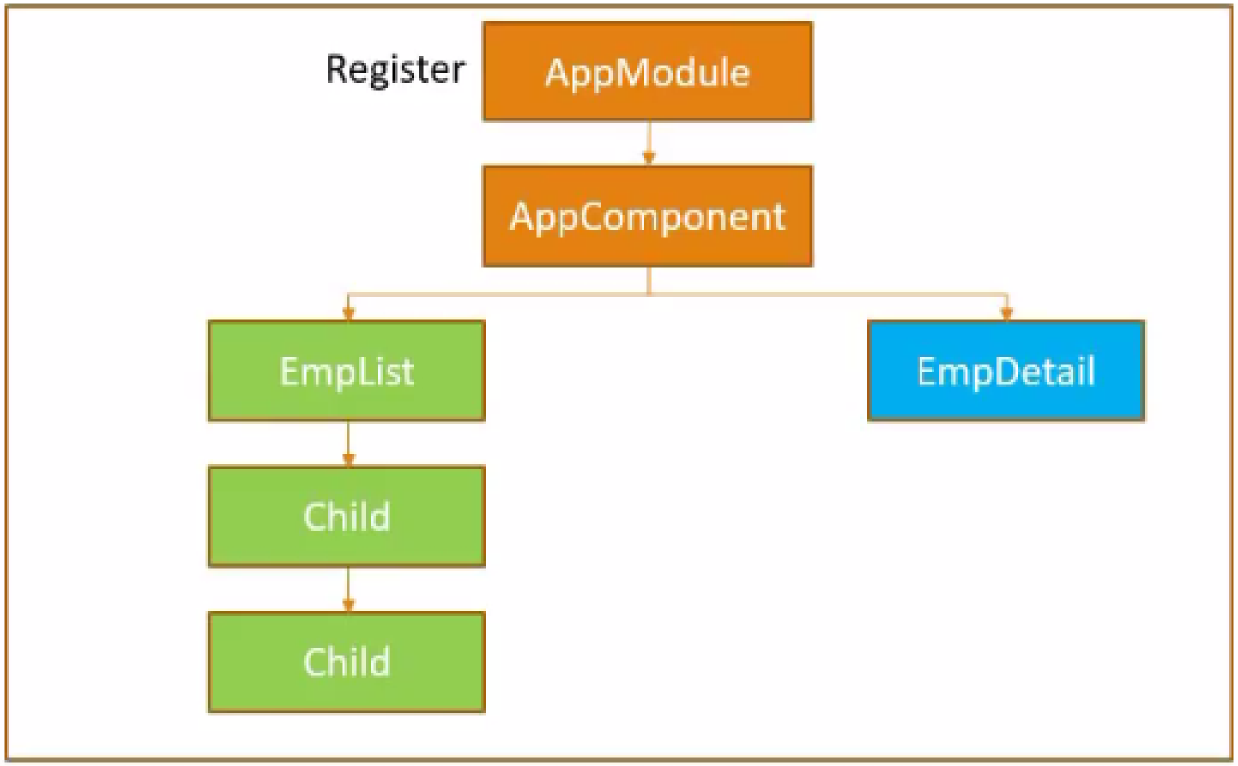
Services(cont.)

- Then create a function inside a service for example to retrieve some date.

```
getEmployees()
{
    return [
        {"name":"Abanoub Nabil" , "age":25},
        {"name":"Abanoub Magdy" , "age":24},
        {"name":"Gerges Abd elmalak" , "age":28},
        {"name":"Rewan Atef" , "age":23},
        {"name":"Salwa Ezz" , "age":22}
            ]
}
```

Services(cont.)

- 2-Register with injector. Where????



Services(cont.)

- In **app.module.ts** include the service in **providers** array and make sure you import it.

```
import { EmployeeService } from
'./Services/employee.service';


providers: [EmployeeService]
```

Services(cont.)

3-List the dependency in the class that needs it.

Steps:

A-Go to the class that want to use this service.

B-In the constructor of the class type this syntax

```
constructor(private
getEmployee:EmployeeServiceService)
```

Services(cont.)

C-Inside ngOninit method and call the method from the service.

```
ngOnInit() {
this.employees=this._employeeservice.getEmployees();
}
```

References

https://angular.io/

Questions

Any Questions?

THANK YOU ☺