

# Services & Dependency Injection

---

UI-Angular



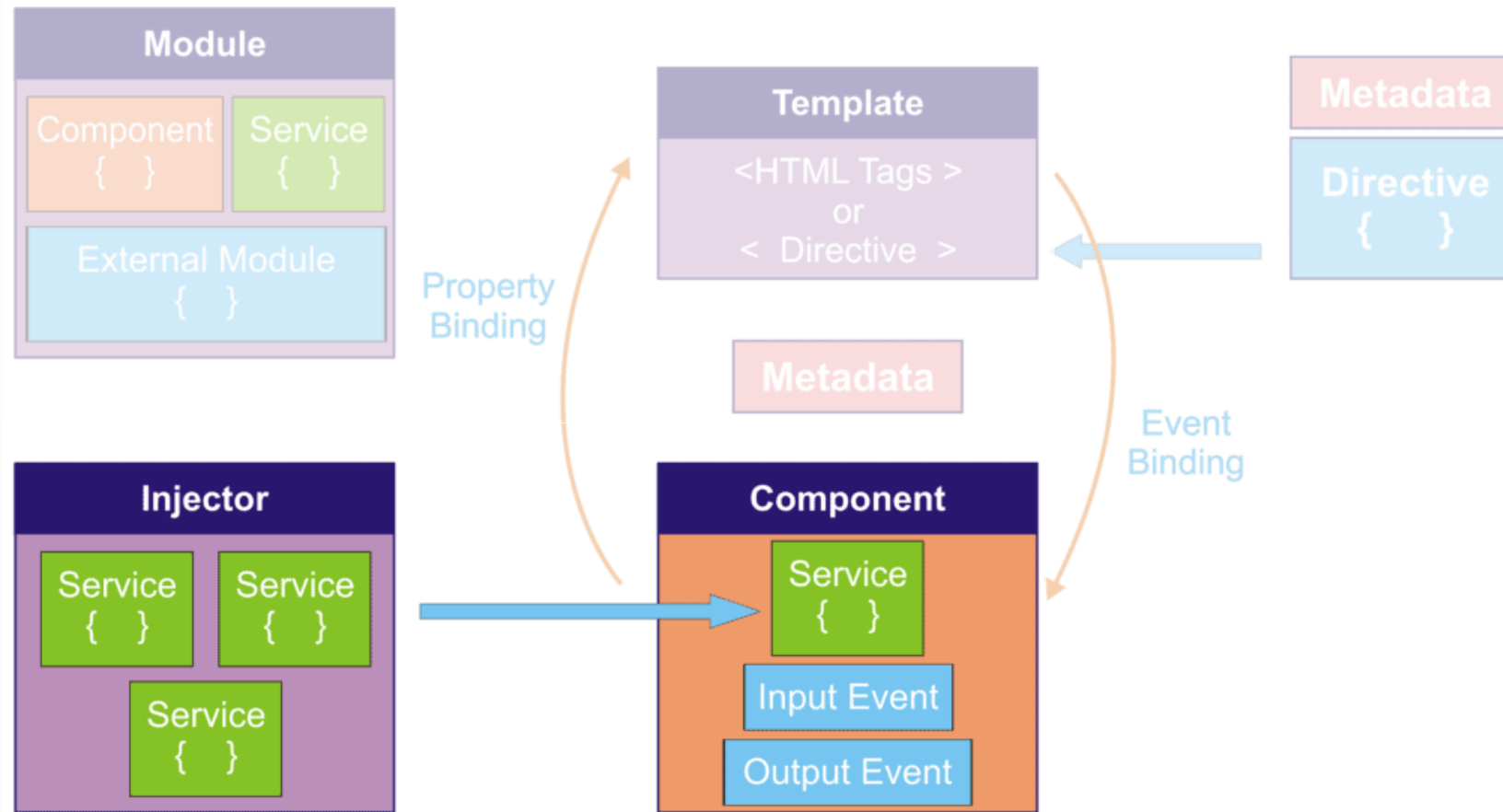
# מה נלמד היום?

---

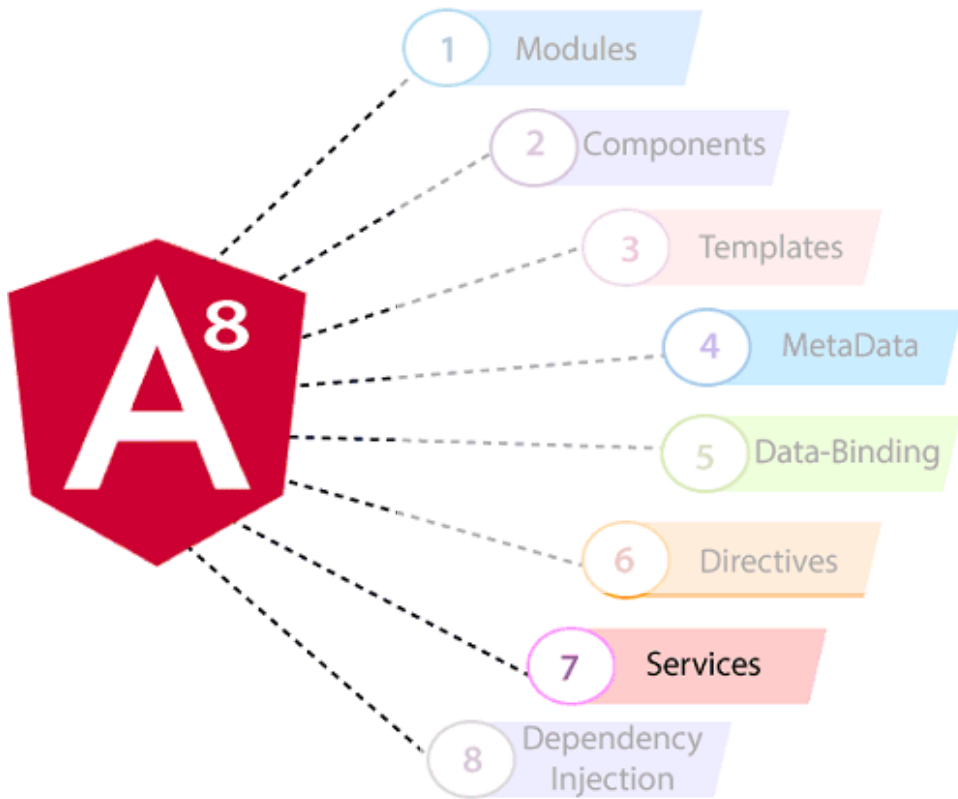
- Services
- Dependency injection
  - Injector, @Injectable
- Component Life Cycle Hook



# ארכיטקטורת יישום Angular



# ארכיטקטורת יישום Angular



■ Service זו בדרך כלל מחלקה שמספקת שירות כלשהו המשותפת לכמה רכיבים באפליקציית angular.



■ המחלקה תכיל לוגיקה כלשהי המתאימה לקומפוננטה אחת או יותר.

# Service

---

- קטע קוד שניתן לבצע בו שימוש חוזר, בחלקים רבים ביישום האנגולר

- מטרות:

- כלליות, אי תלות בקומפוננטות ספציפיות
- שיתוף לוגיקה או נתונים בין קומפוננטות
- הגבלת גישה למשאב/מידע מסוים



## יצירת Service

- בדומה לקומפוננטות ולהוראות Directive ניתן ליצור service בשתי דרכים:
- יצירה ידנית של קובץ ה-service
  - שימוש בפקודות Angular CLI (יודגם בהמשך)



## יצירת Service ידנית

---



## יצירת Service ידנית

src/app/product.ts

```
export class Product {
```

```
  productID : number
```

```
  name : string
```

```
  price : number
```

```
  constructor(productID : number, name : string, price : number) {
```

```
    this.productID = productID
```

```
    this.name = name
```

```
    this.price = price
```

```
  }
```

```
}
```

- לצורך הדוגמה, ניצור תחילה קובץ ts שבו מחלקה כלשהי המחזיקה מידע כלשהו



## יצירת Service ידנית

src/app/product.service.ts

```
import {Product} from './Product'

export class ProductService {

  getProducts() {
    let products:Product[]

    products=[
      new Product(1,'Memory Card',500),
      new Product(2,'Pen Drive',750),
      new Product(3,'Power Bank',100)
    ]
    return products
  }
}
```

- ניצור קובץ מתאים בסיומת service.ts המיועד לשרת את הקומפוננטה הראשית

app.component.ts

```
import { Component } from '@angular/core'
import { ProductService } from './product.service'
import { Product } from './product'

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
})

export class AppComponent
{
  products : Product[]
  productService : ProductService

  constructor(){
    this.productService = new ProductService()
  }

  getProducts() {
    this.products = this.productService.getProducts()
  }
}
```

## יצירת Service ידנית

- נשתמש ב-service שיצרנו קודם לכן, בתוך הקומפוננטה

app.component.html

```
<button (click)="getProducts()">Get Products</button>

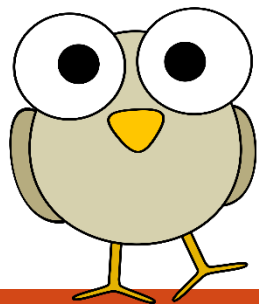
<div class='table-responsive'>
  <table class='table'>
    <thead>
      <tr>
        <th>ID</th>
        <th>Name</th>
        <th>Price</th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let product of products;">
        <td>{{product.productID}}</td>
        <td>{{product.name}}</td>
        <td>{{product.price}}</td>
      </tr>
    </tbody>
  </table>
</div>
```

## בעייתיות ה-Service

- נשים לב שיצרנו מופע של ה-service בתוך הבנאי של מחלקת הקומפוננטה באופן הבא:

```
constructor(){  
    this.productService = new ProductService()  
}
```

- לשיטה זו, חסרונות רבים:
  - כל שינוי במחלקת ה-service מצריך שינוי במחלקת הקומפוננטה
  - אם נרצה להחליף את מחלקת ה-service במחלקת service אחרת נצטרך לשנות בכל רחבי יישום האנגולר
  - קשה לבצע unit-testing (בדיקות) על מחלקת ה-service



??

## בעייתיות ה-Service

- נדגים את הבעייתיות:

- מופע מחלקת השירות נוצר בתוך הבנאי של מחלקת הקומפוננטה

```
constructor(){  
  this.productService = new ProductService()  
}
```

- אם כעת נצטרך לבצע שינוי ומחלקת השירות צריכה לקבל פרמטרים, למשל, כמות המוצרים.  
אז נצטרך לשנות את הבנאי באופן הבא:

```
constructor(num : number){  
  this.productService = new ProductService(num)  
}
```

- זה המקרה הפשוט, אך מה אם השתמשנו במחלקת השירות גם בקומפוננטות אחרות?  
נצטרך לבצע שינויים בכל המקומות – עבודה לא נכונה כי יש תלות במחלקת השירות!

## בעייתיות ה-Service

■ אז מה בכל זאת עושים? נרצה לבטל את התלות

■ במקום לכתוב כך:

```
constructor(){  
    this.productService = new ProductService()  
}
```

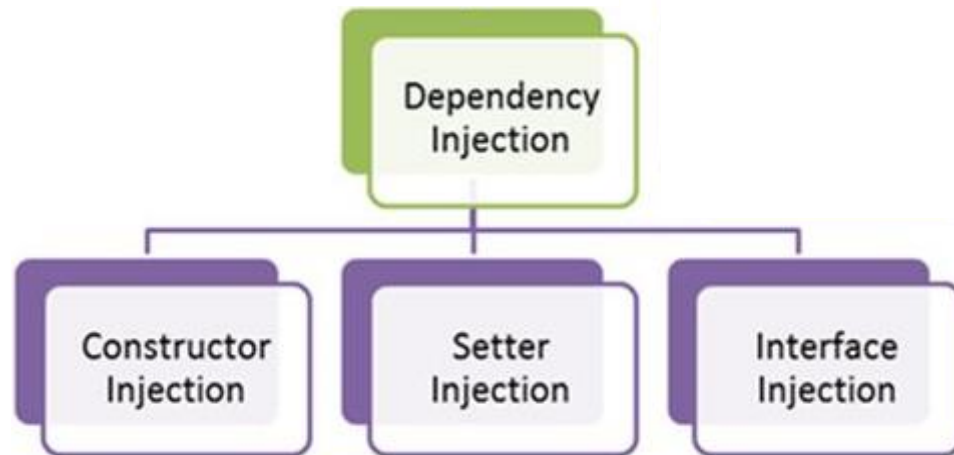
■ נכתוב כך:

```
constructor(productService : ProductService) {  
  
}
```

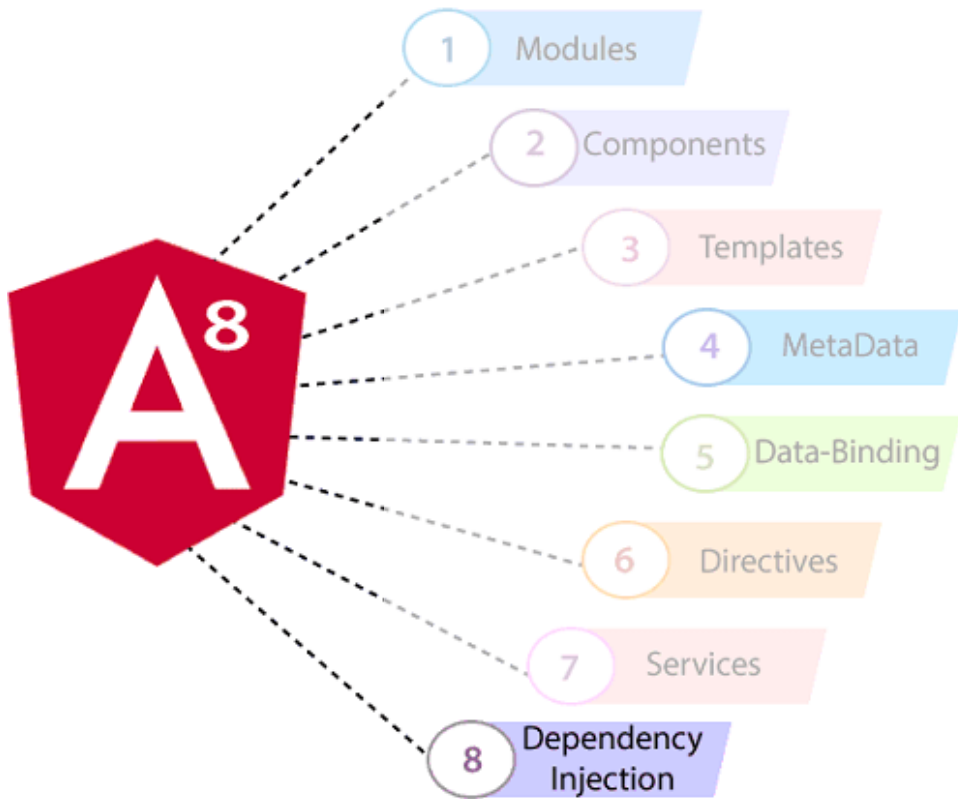
באופן הזה אנחנו מצפים שהבנאי יקבל מופע של מחלקת השירות ולא יצור אותו.  
כעת אין לנו תלות בשינויים המתרחשים במחלקת השירות!

## בעייתיות ה-Service

- השיטה הנ"ל היא תבנית עיצוב (Design Pattern) הנקראת הזרקת תלויות (Dependency injection)



# ארכיטקטורת יישום Angular



- Dependency Injection היא תבנית עיצוב המאפשרת להפוך את הקוד לדינאמי ויעיל.
- מאפשרת לבדוק את הקוד שלנו ביעילות רבה ולא מצריכה שינויים רבים בקוד כאשר מתבצע שינוי בשירות כלשהו.
- מאפשרת יכולת להוסיף את הפונקציונליות של הרכיבים בזמן הריצה.



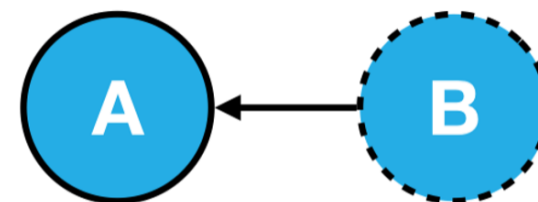
# Dependency Injection

- מהי תלות?
- כאשר קומפוננטה A צריכה את קומפוננטה B כדי לבצע פעולות כלשהן, אזי קומפוננטה B היא תלות של קומפוננטה A

```
export class AppComponent
{
  products : Product[]
  productService : ProductService

  constructor(productService : ProductService) { }

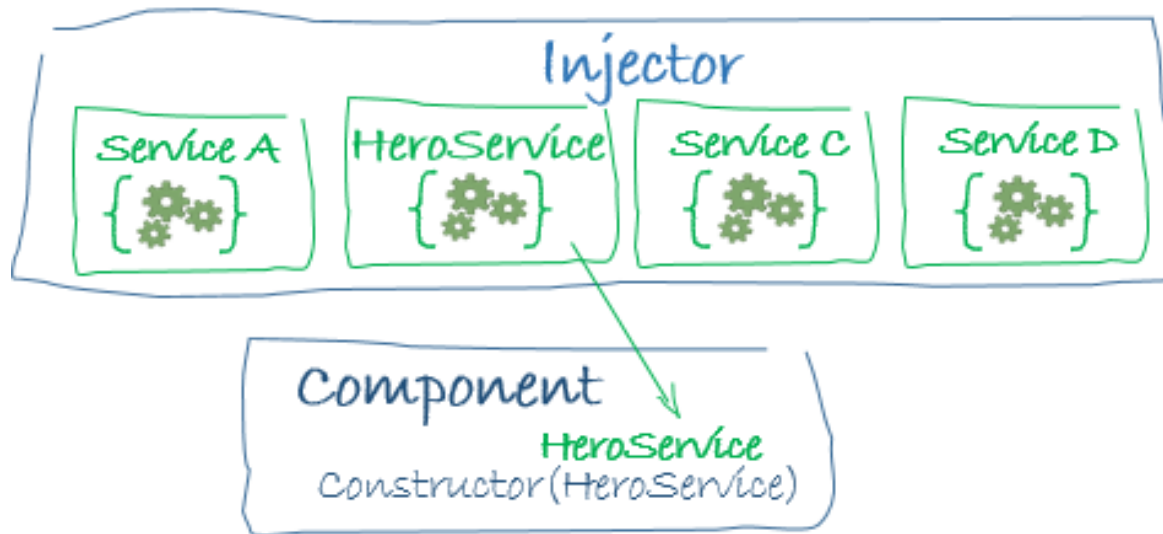
  getProducts() {
    this.products = this.productService.getProducts()
  }
}
```



## הגדרת DI ביישום אנגולר

שלבים:

- יצירת ה-service
- הגדרת התלויות בקומפוננטות




# Dependency Injection

## Angular CLI יצירת service

ניצור service חדש בשם employee ע"י הפקודה:

```
> ng g s employee
```

```
PS C:\Users\morac\angularProjects\my-app> ng g s employee  
CREATE src/app/employee.service.spec.ts (367 bytes)  
CREATE src/app/employee.service.ts (137 bytes)
```



```
TS employee.service.spec.ts    U  
TS employee.service.ts        U
```

```
TS employee.service.ts X  
src > app > TS employee.service.ts > ...  
1  import { Injectable } from '@angular/core';  
2  
3  @Injectable({  
4    providedIn: 'root'  
5  })  
6  export class EmployeeService {  
7  
8    constructor() { }  
9  }
```

# Dependency Injection

## Angular CLI יצירת service

- סימון המחלקה כשירות שניתן להזריק למחלקות אחרות

```
TS employee.service.ts X
src > app > TS employee.service.ts > ...
1  import { Injectable } from '@angular/core';
2
3  @Injectable({
4    providedIn: 'root'
5  })
6  export class EmployeeService {
7
8    constructor() { }
9  }
```

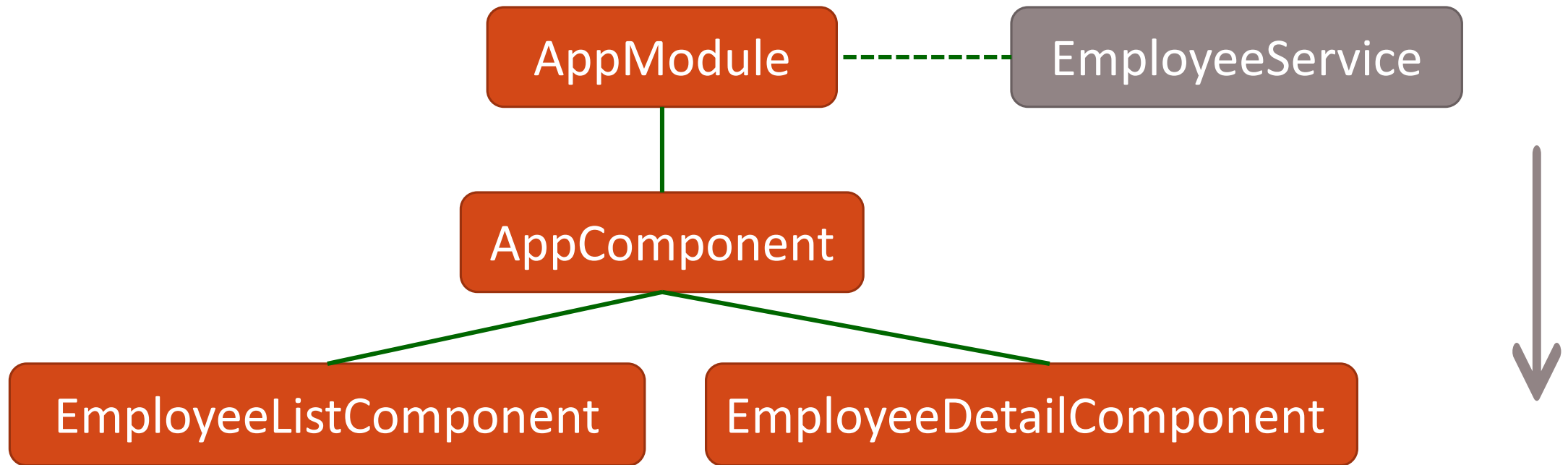
# Dependency Injection

## Angular CLI יצירת service

```
TS employee.service.ts X
src > app > TS employee.service.ts > ...
1  import { Injectable } from '@angular/core';
2
3  @Injectable({
4    providedIn: 'root'
5  })
6  export class EmployeeService {
7
8    constructor() { }
9  }
```

- הגדרת רמת השיתוף של מחלקת השירות
- 'root' כל מה שנמצא תחת AppModule

## רמת השיתוף



# Dependency Injection

## Angular CLI יצירת service

TS employee.service.ts X

src > app > TS employee.service.ts > ...

```
1  import { Injectable } from '@angular/core';
2
3  @Injectable({
4    providedIn: 'root'
5  })
6  export class EmployeeService {
7
8    constructor() { }
9
10   getEmployees(){
11     return [
12       {id: 1, name: "Idan Guy", age: 25},
13       {id: 2, name: "Sharon Tal", age: 30},
14       {id: 3, name: "Yuval Shir", age: 18},
15       {id: 4, name: "Adam Gil", age: 22},
16     ]
17   }
18 }
19 }
```

- נכניס למחלקה מידע שיכול לשמש כמה קומפוננטות

# Dependency Injection

## הגדרת התלויות בקומפוננטות

```
TS employee-list.component.ts X
src > app > employee-list > TS employee-list.component.ts > ...
1 import { Component, OnInit } from '@angular/core';
2 import { EmployeeService } from '../employee.service';
3
4 @Component({
5   selector: 'app-employee-list',
6   template: `
7     <h1> Employees List</h1>
8     <ul>
9       <li *ngFor="let employee of employees">{{employee.name}}</li>
10     </ul>
11   `,
12   styleUrls: ['./employee-list.component.css']
13 })
14 export class EmployeeListComponent implements OnInit {
15   employees = []
16   constructor(private employeeService : EmployeeService) { }
17   ngOnInit(): void {
18     this.employees = this.employeeService.getEmployees()
19   }
20 }
21
22
23
```

```
TS employee-detail.component.ts X
src > app > employee-detail > TS employee-detail.component.ts > ...
1 import { Component, OnInit } from '@angular/core';
2 import { EmployeeService } from '../employee.service';
3
4 @Component({
5   selector: 'app-employee-detail',
6   template: `
7     <h1> Employees Detail</h1>
8     <ul>
9       <li *ngFor="let employee of employees">
10         <b> id: </b> {{employee.id}}
11         <b> name:</b> {{employee.name}}
12         <b> age: </b> {{employee.age}}
13       </li>
14     </ul>
15   `,
16   styleUrls: ['./employee-detail.component.css']
17 })
18 export class EmployeeDetailComponent implements OnInit {
19   employees = []
20   constructor(private employeeService : EmployeeService) { }
21   ngOnInit(): void {
22     this.employees = this.employeeService.getEmployees()
23   }
24 }
25
26
27
```



## הגדרת התלויות בקומפוננטות

```
app.component.html X
src > app > app.component.html > ...
1 <app-employee-list></app-employee-list>
2 <app-employee-detail></app-employee-detail>
```

### Employees List

- Idan Guy
- Sharon Tal
- Yuval Shir
- Adam Gil

### Employees Detail

- **id:** 1 **name:** Idan Guy **age:** 25
- **id:** 2 **name:** Sharon Tal **age:** 30
- **id:** 3 **name:** Yuval Shir **age:** 18
- **id:** 4 **name:** Adam Gil **age:** 22

# Component Life Cycle Hook

---

כאשר יישום אנגולר מתחיל לרוץ הוא מבצע מספר פעולות:

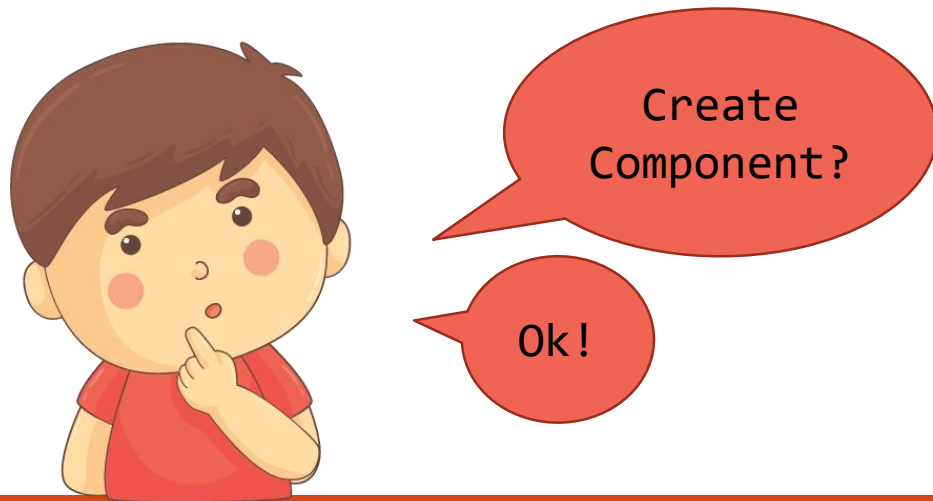
- יוצר ומעבד את הקומפוננטה הראשית וכל הקומפוננטות שתחתיה
- עיבוד תהליכי Data Binding אם קיימים
- עיבוד תקשורת בין הקומפוננטות אם קיימת
- עיבוד Content Projection אם קיים
- מחיקת קומפוננטות כאשר אין בהן צורך יותר

# Component Life Cycle Hook

---

כיצד יישום אנגולר יידע שצריך לטפל בכל אלה?

- Angular life cycle hooks
- אירועים שמתרחשים ואנגולר מגיבה אליהם באמצעות פונקציות מוגדרות מראש



דוגמה שראינו בעבר:

■ `ngOnChanges`

# Component Life Cycle Hook

```
import { Component, Input, OnChanges, SimpleChanges, SimpleChange } from '@angular/core';
```

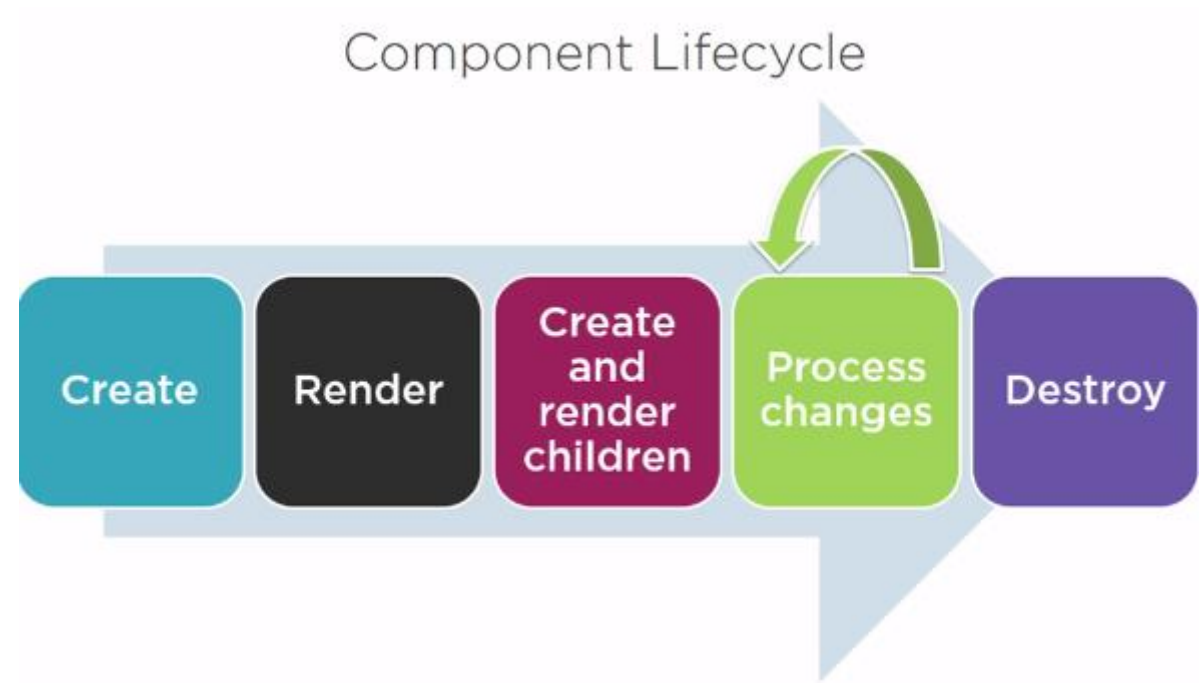
```
@Component({  
  selector: 'child-component',  
  template: `<h2>Child Component</h2>  
             current count is {{ count }}`  
})  
export class ChildComponent implements OnChanges {  
  @Input() count: number;
```

```
    ngOnChanges(changes: SimpleChanges) {  
      for (let property in changes) {  
        if (property === 'count') {  
          console.log('Previous:', changes[property].previousValue)  
          console.log('Current:', changes[property].currentValue)  
          console.log('firstChange:', changes[property].firstChange)  
        }  
      }  
    }  
  }  
}
```

יש  
שינוי?

# Component Life Cycle Hook

- ngOnChanges
- ngOnInit
- ngDoCheck
- ngAfterContentInit
- ngAfterContentChecked
- ngAfterViewInit
- ngAfterViewChecked
- ngOnDestroy



<https://angular.io/guide/lifecycle-hooks>

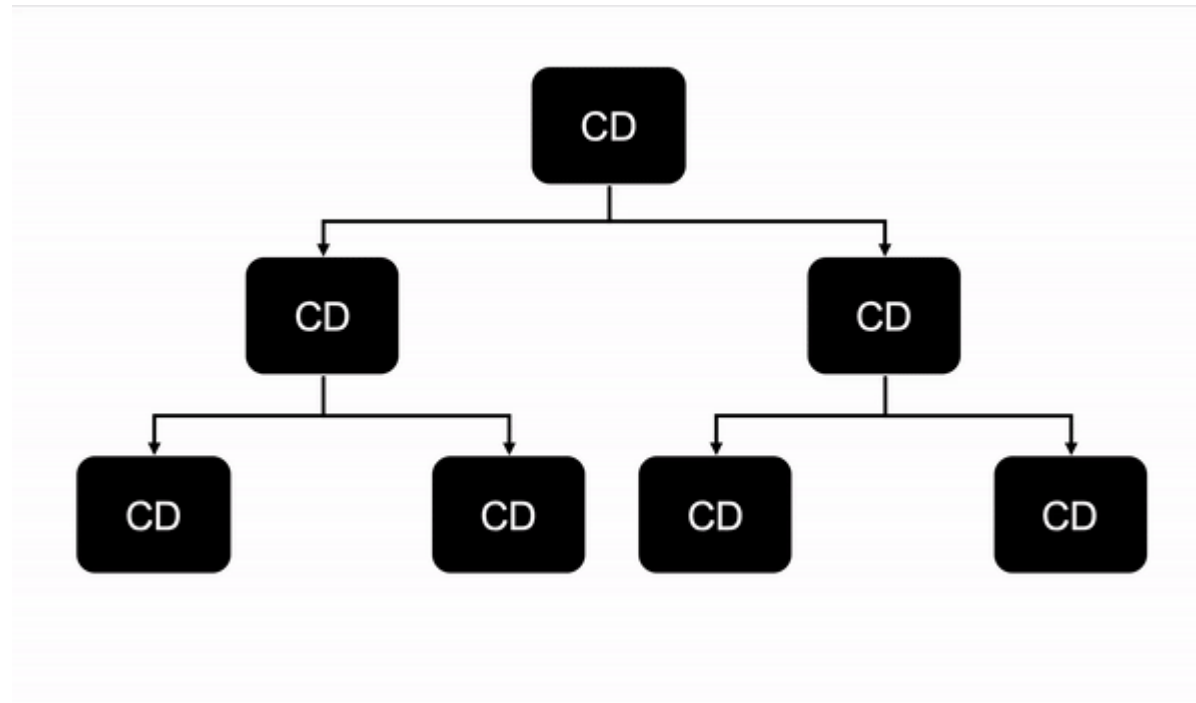
## Change Detection

- תהליך עדכון התצוגה (DOM) כאשר הנתונים משתנים
- הסנכרון בין HTML למידע הפנימי ביישום

האלגוריתם:

- מתרחש שינוי כלשהו ב-model כלומר באחת או יותר מהקומפוננטות
- **אנגולר מזהה את השינוי ע"י אירוע**
- כל הקומפוננטות ביישום נסרקות לזיהוי השינוי
- אם יש שינוי בקומפוננטה כלשהי, ה-view שלה יעודכן

## Change Detection



## Constructor

- כאשר אנגולר יוצרת קומפוננטה הפונקציה הראשונה שנקראת היא הבנאי
- אנגולר עושה שימוש בבנאי:
  - ליצירת משתנים ואתחולם
  - להזרקת תלויות





## ngOnChanges

- כאשר אנחנו רוצים להעביר מידע מקומפוננטה האב לקומפוננטה הבן אנחנו משתמשים ב-@Input
- כאשר מתרחש שינוי באחד מהמשתנים המוצמדים ב-@Input הפונקציה ngOnChanges נקראת באופן אוטומטי.

```
interface OnChanges {  
  ngOnChanges (changes: SimpleChanges): void  
}
```

## ngOnInit

- פונקציה חד פעמית
- בדור"כ נציב בפונקציה לוגיקה ובבנאי מידע
- הפונקציה מופעלת לאחר שאנגולר יצרה את הקומפוננטה ועדכנה את המשתנים המוגדרים ב-@Input
- אחרי ה-constructor
- אחרי ngOnChanges

```
interface OnInit {  
  ngOnInit(): void  
}
```

## ngAfterContentInit

- הפונקציה נקראת לאחר שהסתיים במלואו תהליך ה-Content Projection

child.component.html

```
<h2> Child Component </h2>  
<ng-content></ng-content>
```

```
interface AfterContentInit {  
  ngAfterContentInit(): void  
}
```

app.component.html

```
<h1> Parent Component </h1>  
<app-child> This <b>content</b> is injected from parent </app-child>
```

# Component Life Cycle Hook

## ngOnDestroy

- הפונקציה נקראת רגע לפני שאנגולר מוחקת קומפוננטה

```
interface OnDestroy {  
  ngOnDestroy(): void  
}
```

```
@Component({  
  selector: 'my-cmp',  
  template: `...`  
})
```

```
class MyComponent implements OnDestroy {  
  ngOnDestroy() {  
    ...  
  }  
}
```

# Component Life Cycle Hook

## דוגמת שימוש ב-CLCH

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

@Component({
  selector: 'app-root',
  template: `
    <h2> Life Cycle Hook </h2>` ,
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {

  constructor() {
    console.log("AppComponent:Constructor");
  }

  ngOnInit() {
    console.log("AppComponent:OnInit");
  }
}
```

- import ל-interface אותו נרצה לממש
- נגדיר שמחלקת הקומפוננטה/directive מממשת את ה-interface
- נממש את הפונקציה המתאימה

# Component Life Cycle Hook

## שימוש

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

@Component({
  selector: 'app-root',
  template: `
    <h2> Life Cycle Hook </h2>` ,
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {

  constructor() {
    console.log("AppComponent:Constructor");
  }

  ngOnInit() {
    console.log("AppComponent:OnInit");
  }
}
```

Console:

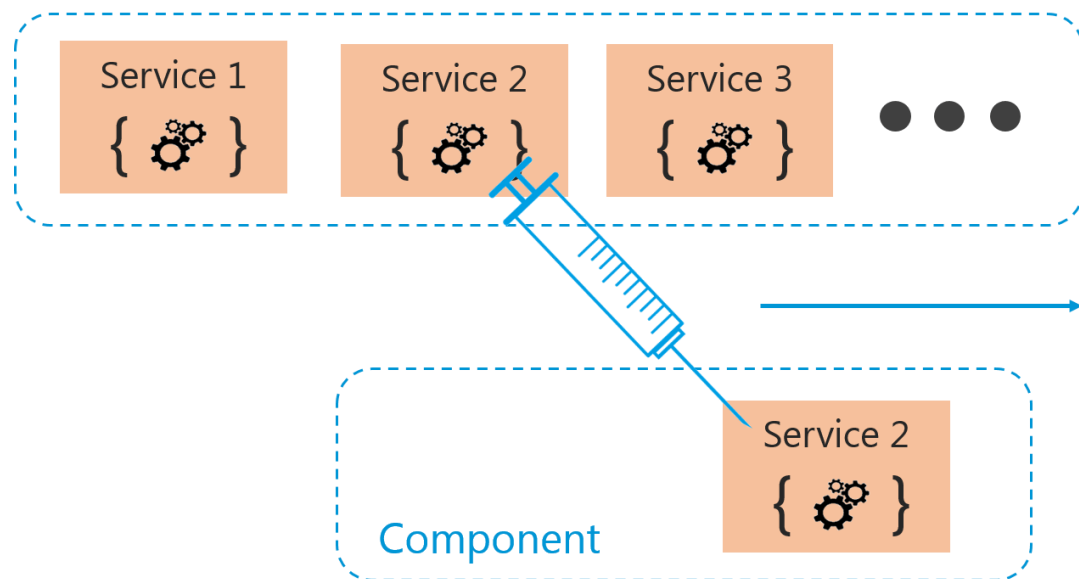
```
AppComponent:Constructor
AppComponent:OnInit
```

## יצירת קומפוננטה

---

1. OnChanges
2. OnInit
3. DoCheck
4. AfterContentInit
5. AfterContentChecked
6. AfterViewInit
7. AfterViewChecked

# יצירת קומפוננטה



**Dependency  
Injection**

