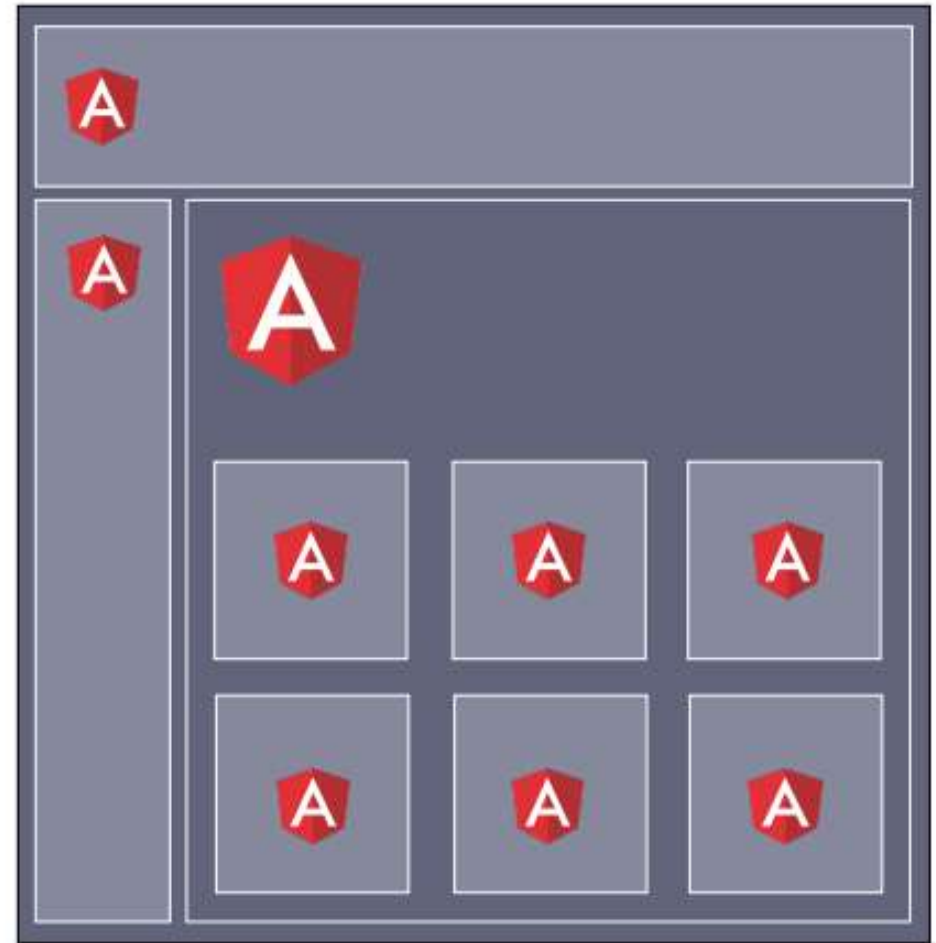


Angular czesc 2

dr inż. Grzegorz Rogus

Aplikacja składa się z komponentów

W Angular aplikacja zbudowana jest z drzewa komponentów. Każdy komponent może mieć zestaw komponentów „dzieci” oraz rodzica. Naszym głównym komponentem jest jego korzeń, tzw. root component.



Połączenie pomiędzy komponentami

```
@Component({  
  selector: "my-child",  
  template: "This is a child component"  
})  
export class ChildComponent {}
```

POTOMEK

Dodaj selektor
komponentu dziecka w
szablonie rodzica

```
@Component({  
  selector: "app",  
  template: "<my-child></my-child>",  
})  
export class AppComponent {}
```

RODZIC

W jaki sposób komponenty się komunikują??

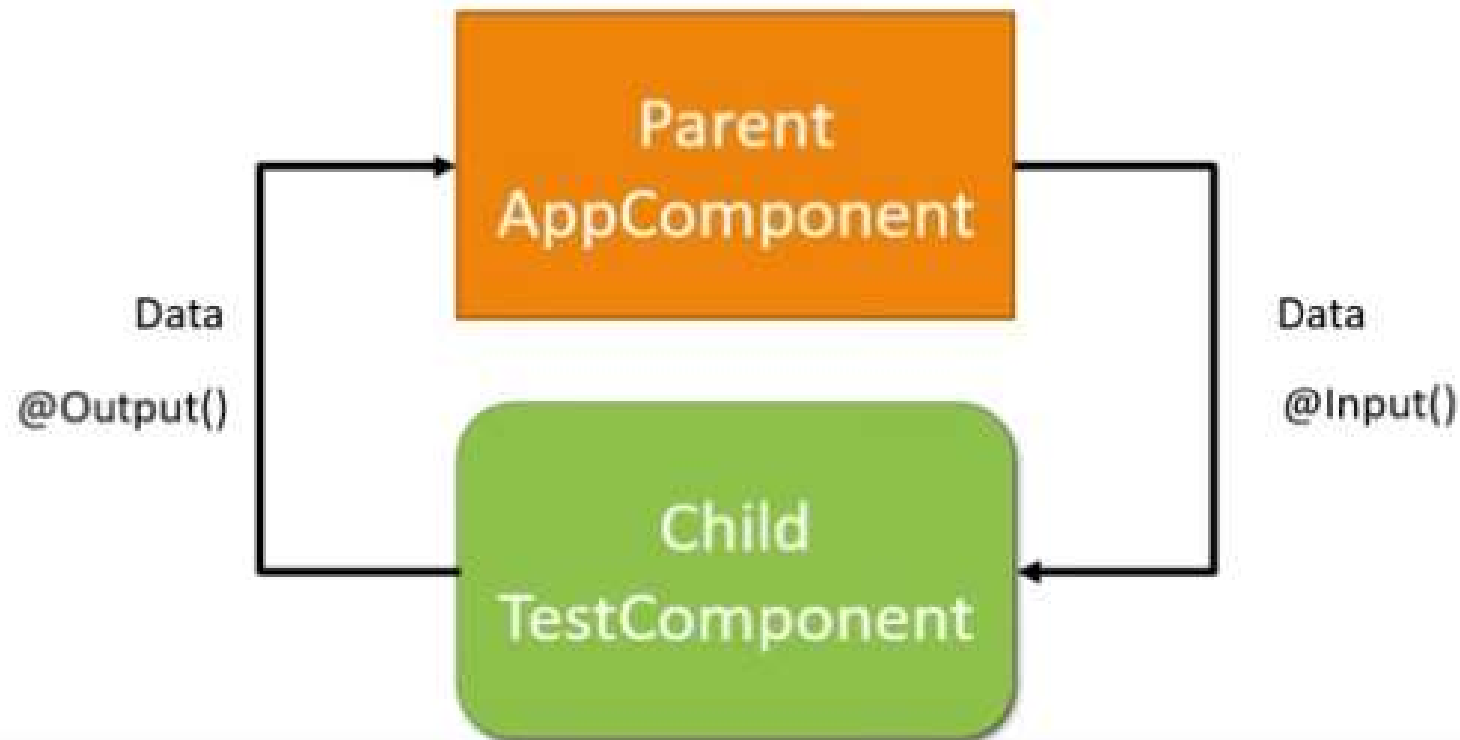
1. Inputs i Outputs (tylko powiązane)

2. Usługi (wszystkie)

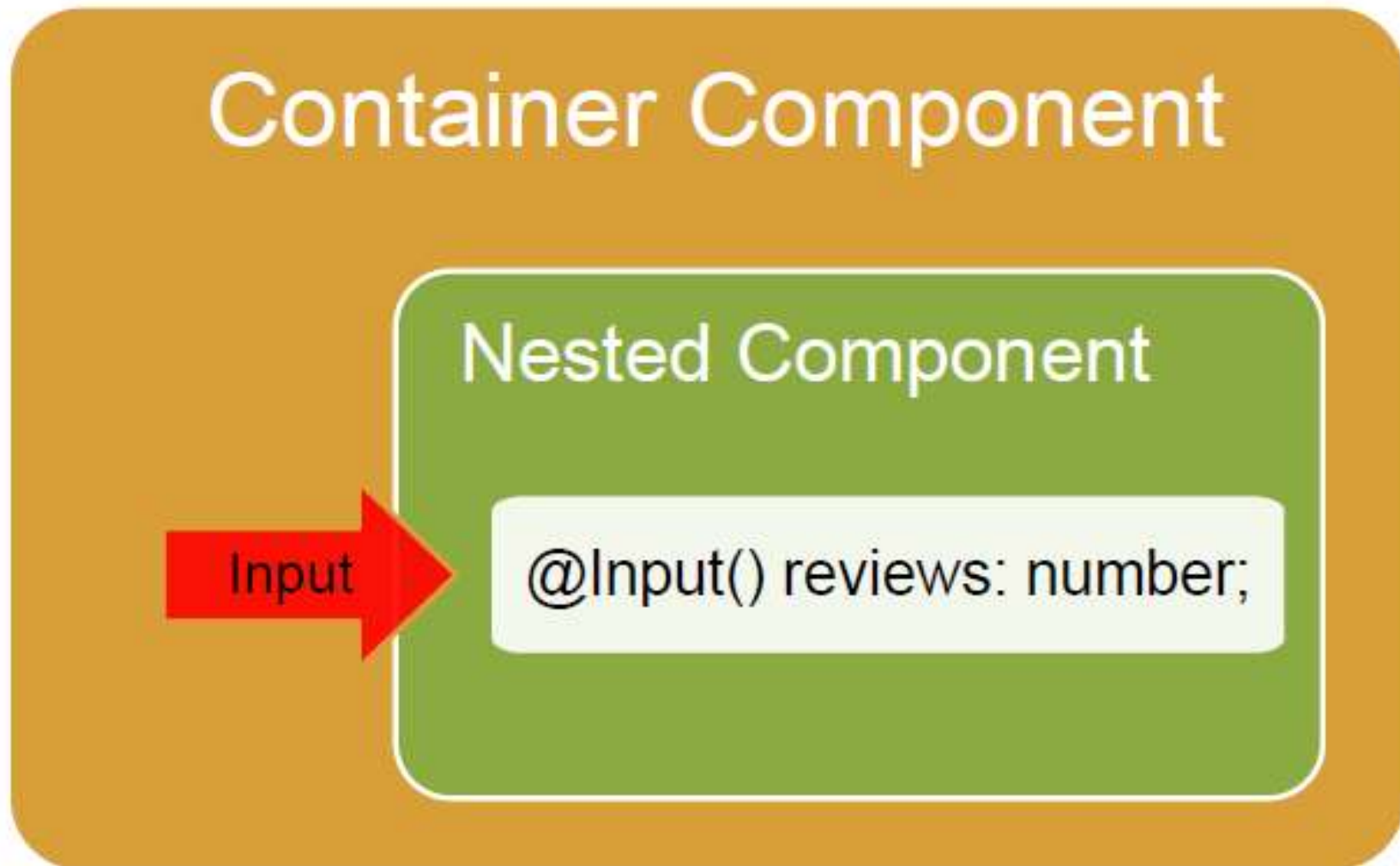
Komunikacja między komponentami



Komunikacja między komponentami



Input



komponent rodzica ma możliwość przekazania do dziecka danych, które mogą determinować zachowanie komponentu lub w ogóle – pozwolić na jego odpowiednie wyrenderowanie. Jest to możliwe dzięki adnotacji `@Input()`

Komponent: komunikacja "do"

```
@Component ({  
  selector: 'test-input',  
  template: '...'  
})  
export class GR_Test {  
  @Input() item: myType;  
}
```

```
// lub:  
@Component ({  
  selector: 'test-input',  
  inputs: ['item'],  
  template: '...'  
})  
export class GR_Test {  
  item: myType;  
}
```

```
// przekazywanie przez zmienną:  
<test-input [item]="myItem-GR"></test-input>  
  
// przekazywanie wartości bezpośrednio  
<test-input item="myItem-GR"></test-input>
```


Krok 1. Dodanie nowej własności typu input



```
import {Component, Input} from "@angular/core";
```


```
@Component({  
  selector: "my-child",  
  template: "This is a child component with a message: {{ message }}"  
})
```

```
export class ChildComponent {  
  @Input() message: string;  
}
```



Krok 2: Powiąż zmienną rodzica z tą własnością

```
@Component({  
  selector: "app",  
  template: `  directives: [ChildComponent]  
})  
  
export class AppComponent {  
  pozdrowienia = "Pozdrowienia od GR";  
}
```



Input

child_component.ts

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

@Component({
  selector: 'child'
})

export class ChildComponent {
  Input() reviews: number;
}
```

child_component.html

```
<div>
  <p>{{ reviews }}</p>
</div>
```

parent_component.ts

```
export class ParentComponent {
  books: any[] = [{
    bookReviews: 15
  }]
}
```

parent_component.html

```
<div><h1>Parent Title</h1>
<p>body text...</p>
<child [reviews]="book.bookReviews">
</child>
</div>
```

Komponent: komunikacja "z" - zdarzenia

// lub:

```
@Component ({  
  selector: 'test-output',  
  template: '...' })  
export class Hello {  
  @Output() completed:  
  EventEmitter<boolean> = new EventEmitter<boolean>();  
}
```

```
<test-output (completed)="saveProgress(item)"></test-output>
```

// alternatywna składnia - dlatego nie prefixujemy zdarzeń "on"

```
<test-output on-completed="saveProgress(item)"></test-output>
```

Implementacja



```
import {Component, Input, EventEmitter, Output} from "@angular/core";

@Component({
  selector: "my-child",
  template: `This is a child component with a message: {{ message }}`
})
export class ChildComponent {
  @Input() message: string;
  @Output() signaledIsHungry = new EventEmitter<string>();
}
```



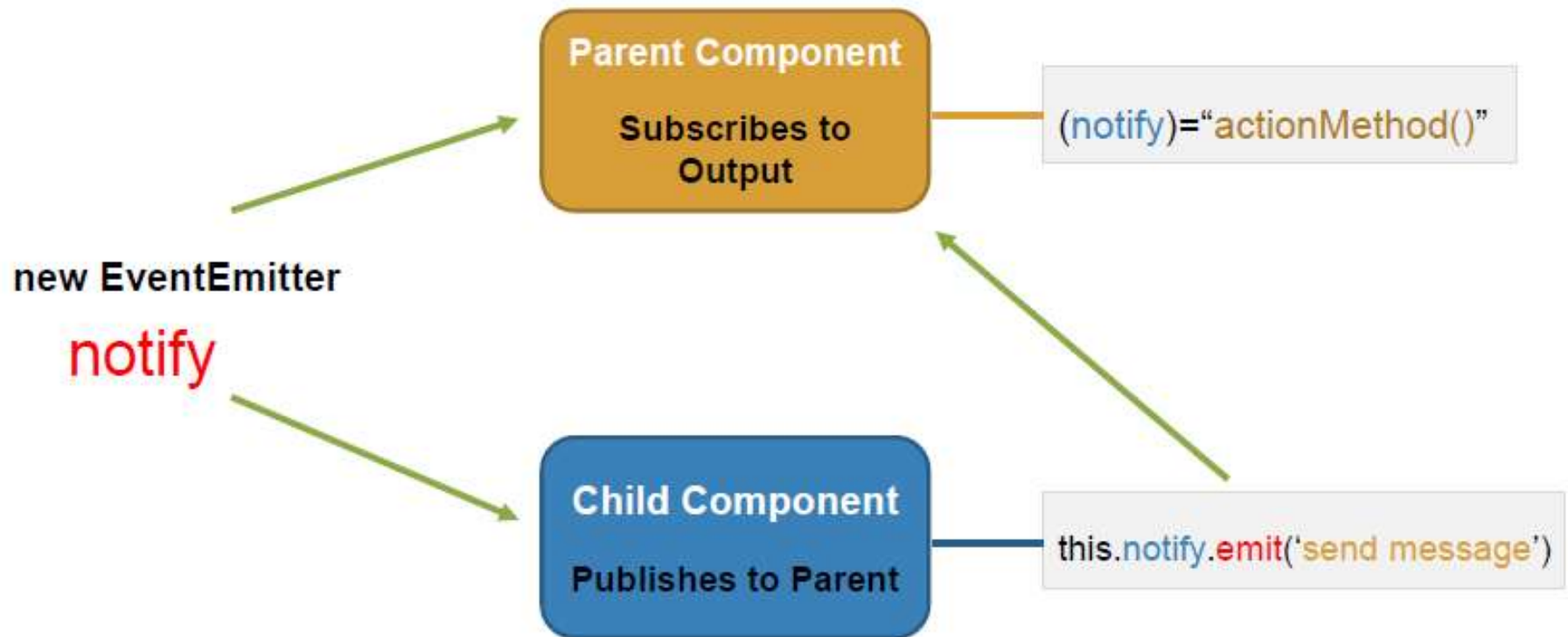
EventEmitter jest klasą generyczną – szablonem.

parametr jaki będzie przekazywał w postaci argumentu, będzie typu **string**.

Dodanie funkcji subskrybujacej

```
@Component({
  selector: "app",
  template: `
```

Subskrypcja zdarzeń



Output

child_component.ts

```
import { Output, EventEmitter } from '@angular/core';

export class ChildComponent {
  Output() notify: EventEmitter<string> =
    new EventEmitter<string>();

  onClick(): void {
    this.notify.emit('Message from child');
  }
}
```

parent_component.ts

```
export class ParentComponent {

  onNotifyClicked(message: string): void {
    this.showMessage = message;
  }
}
```

parent_component.html

```
<div><h1>Parent Title</h1>
<p>{{ showMessage }}</p>
<child (notify)="onNotifyClicked($event)">
</child>
</div>
```

child_component.html

```
<div
  (click)="onClick()">
  <button>Click Me</button> </div>
```




Usługi

Service

Klasa do specyficznych celów:

- dzielenia danych
- Implementacja logiki aplikacyjnej/biznesowej
- Zewnętrzna interakcja – odczyty danych z serwera

Usługi

- Klasy implementujące funkcje potrzebne w aplikacji
 - Pojedyncza usługa powinna odpowiadać za konkretną funkcjonalność
- Podział logiki między klasy komponentów i usług
 - Klasa komponentu powinna zawierać logikę specyficzną dla widoku
 - Właściwości i metody do wiązania danych z widoku
 - Pośredniczenie między widokiem a logiką biznesową (modelem)
 - Klasy usług nie powinny zależeć od widoków
 - Komunikacja z backendem, walidacja, obsługa logu itd.
- Usługi są udostępniane komponentom i innym usługom przez wstrzykiwanie zależności (ang. dependency injection)
 - Dekorator @Injectable oznacza klasy będące usługami oraz klasy i komponenty zależne od usługi
 - Wstrzykiwanie realizowane przez typowany parametr konstruktora
 - Dostawcy usług (ang. providers), typowo klasy usług, rejestrowani na poziomie modułów lub komponentów

Implementacja usług - uwagi

- Klasy komponentów powinny być szczupłe, bez nadmiaru (kodu, funkcjonalności). Nie pobierają danych z serwera, walidują danych wejściowych od użytkownika czy zapisują logi bezpośrednio do konsoli. Takie zadania są delegowane do usług.
- **Zadaniem komponentu jest udostępnienie użytkownikowi danej funkcjonalności i nic ponad to.**
- Komponent pośredniczy pomiędzy widokiem (renderowanym na bazie szablonu) i logiką aplikacji (która często zawiera jakąś wiedzę o modelu).
- Dobry komponent prezentuje właściwości i metody do wiązania danych. Wszystkie nietrywialne zadania są delegowane do usług.
- Angular pomaga nam *przestrzegać* tych wytycznych poprzez ułatwianie nam podzielenia logiki aplikacji na usługi i uczynienie tych usług dostępnymi w komponentach poprzez *wstrzykiwanie zależności*

Wstrzykiwanie zależności

Wstrzykiwanie zależności:

- jest sposobem na dostarczenie nowej instancji klasy razem ze wszystkimi zależnościami, które są jej potrzebne.
- większość zależności to usługi.



```
Component
{
  Constructor(service)
}
```

A hand-drawn diagram showing a class named 'Component' with a constructor that takes a 'service' parameter. The word 'service' is written in green and underlined with three lines.

Angular:

- używa wstrzykiwania zależności w celu dostarczenia nowych komponentów razem z usługami, których potrzebują,
- rozpoznaje, których usług potrzebuje komponent sprawdzając na typy parametrów jego konstruktora.

Usługi - service

- Generacja automatyczna ng -g s nazwaUsługi

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

@Injectable()
export class CountryService {

  constructor() { }

  getCountry() {
    return ["Polska", "niemcy", "Rosja", "czechy"];
  }

}
```

```
import { CountryService } from '../country.service';
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-nowy-komponent',
  templateUrl: './nowy-komponent.component.html',
  styleUrls: ['./nowy-komponent.component.css']
})
export class NowyKomponentComponent implements OnInit {

  buttonStatus = true;
  kraje;

  constructor() {
    let service = new CountryService();
    this.kraje = service.getCountry();
  }

}
```

Usługa z DI i bez - porównanie

Without DI

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

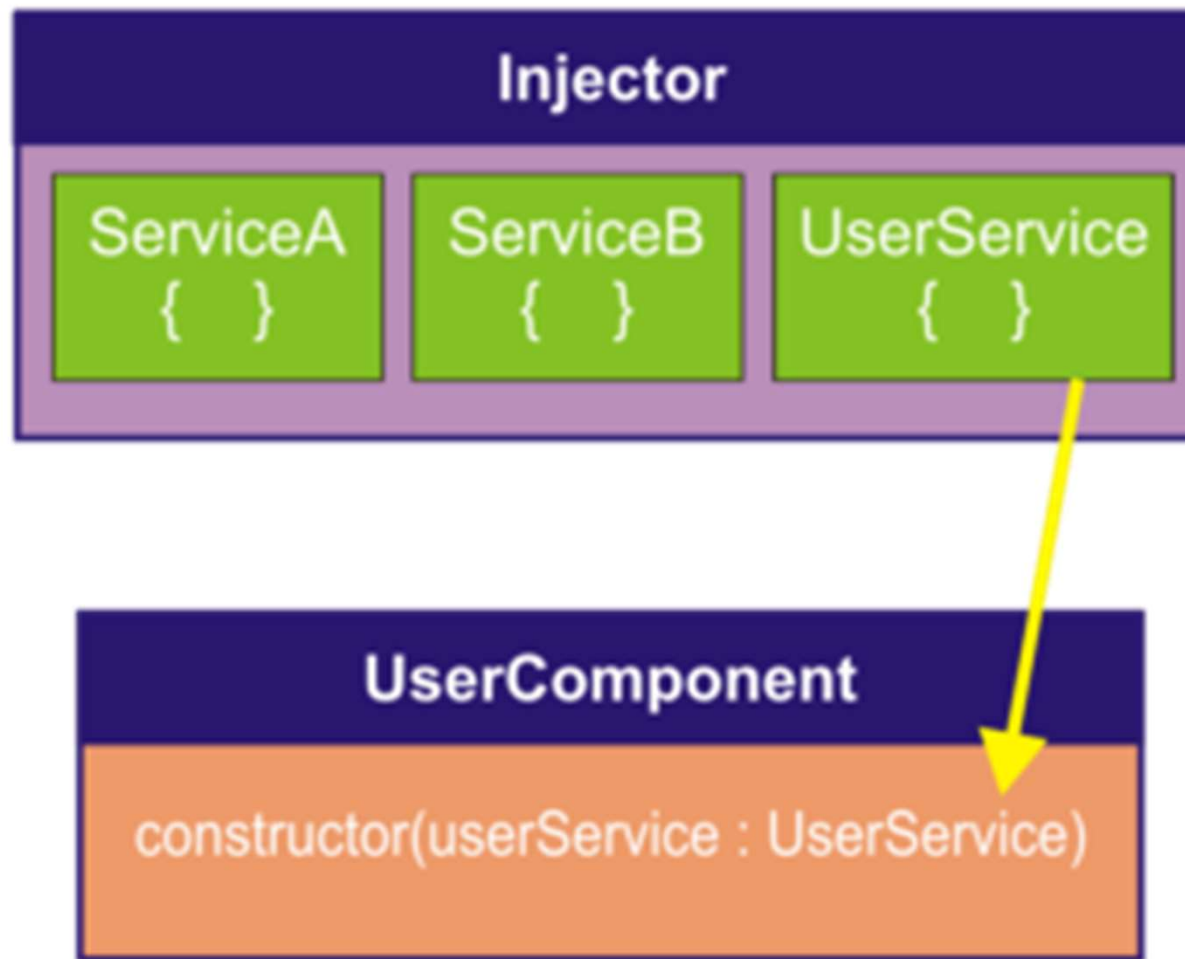
With DI

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

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

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

Usługa – wstrzykiwanie zależności



Wstrzykiwanie zależności

Klasę rozszerzoną dekoratorem (*np.* @Injectable)

```
@Injectable()  
class TodosService {  
    constructor() {}  
}
```

... i zarejestrowaną jako *provider*

```
@NgModule({  
    providers: [  
        TodoService,
```

... możemy wstrzyknąć do konstruktorów innych klas

```
class TodoItemComponent {  
    constructor(ts: TodoService) {} // skrócony zapis  
    // constructor(@Inject(TodosService) ts) {} // pełen zapis  
}
```


Usługa i wstrzykiwanie zależności – przykład

Usługa

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Injectable()
export class StudentService {

  constructor(private http: HttpClient) { }

  ...
}
```

Rejestracja dostawcy usług w module

```
@NgModule({
  declarations: [
    AppComponent,
    StudentsComponent, ...],
  imports: [...],
  providers: [StudentService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Wstrzyknięcie usługi w komponencie

```
import { Component, OnInit } from '@angular/core';
import { StudentService } from '../student.service';

@Component({
  selector: 'app-students', templateUrl: './students.component.html', styleUrls: ['./students.component.css']
})
export class StudentsComponent implements OnInit {

  constructor(private studentService: StudentService) { }

  ...
}
```

Wstrzykiwanie do konstruktora – Dependency Injection

```
import { Component } from '@angular/core';
import { KsiazkiService } from '../ksiazki.service';
```

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
```

```
export class AppComponent {
  name = 'Grzegorz Rogus';
```

```
  ksiazki;
```

```
  constructor(service: KsiazkiService) {
    this.ksiazki = service.getKsiazki();
  };
}
```

```
<div style="text-align:center">
```

```
  <h1>
```

```
    Witaj {{name}}!
```

```
  </h1>
```

```
</div>
```

```
<p>
```

```
  Ksiazki warté polecenia na temat technologii Webowych:
```

```
</p>
```

```
<ul>
```

```
  <li *ngFor=" let ksiazka of ksiazki">
```

```
    {{ksiazka.title}} w cenie {{ksiazka.price}}
```

```
  </li>
```

```
</ul>
```

Wstrzykiwanie serwisu do Komponentu



Provider



Injector

```
export class DashboardComponent  
  constructor(private dataService: DataService) { }  
}
```

Wstrzykiwanie zależności Dependency Injection

```
constructor() {  
  let service = new CoursesService();  
  this.courses = service.getCourses();  
}
```

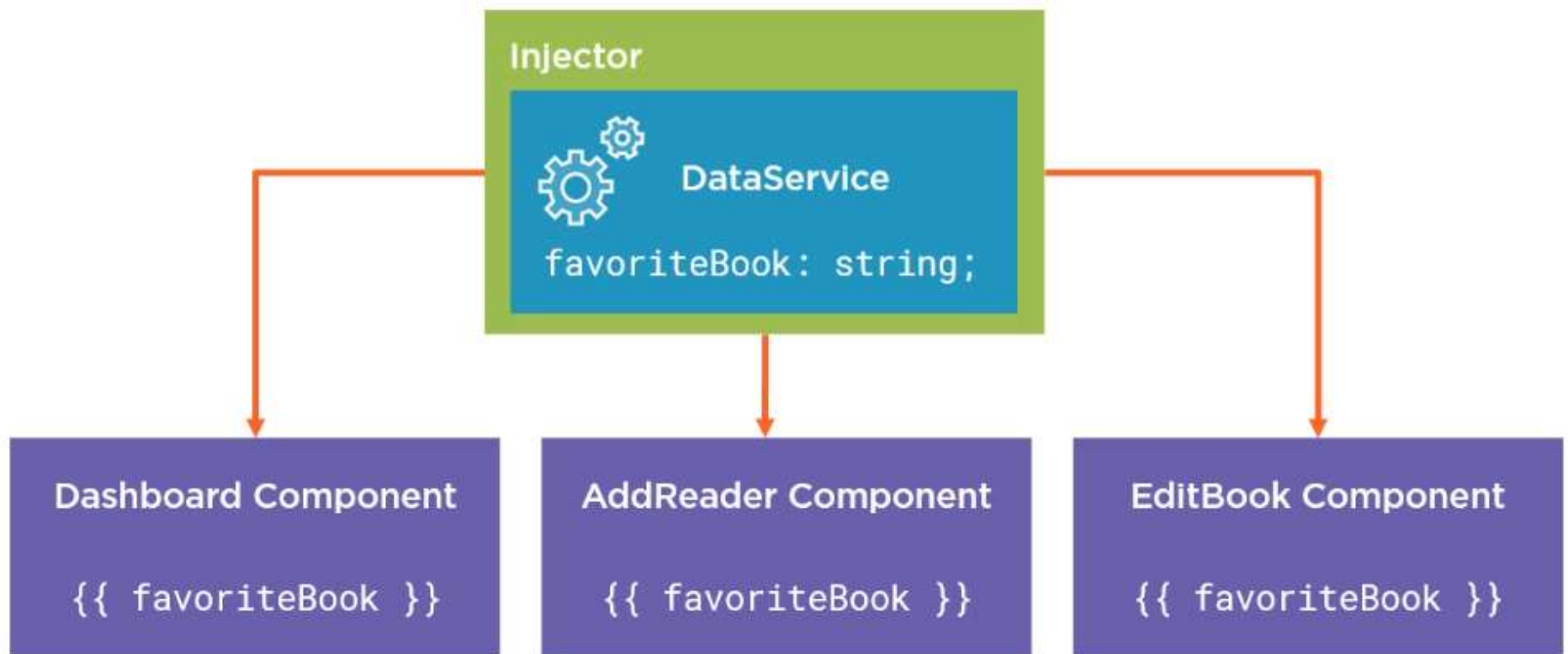
ZLE

VS

```
constructor(service: CoursesService) {  
  this.courses = service.getCourses();  
}
```

DOBRZE

Wymiana danych poprzez usługę



Model dziedziny i Mock data

Dobrą praktyką projektową jest wyizolowanie struktury danych oraz ewentualnych danych od komponentu.

```
export interface Product {  
  id: number;  
  modelName: string;  
  color: string;  
  productType: string;  
  brand: string;  
  price: number;  
}
```

product.ts

```
import { Product } from '../models/product';  
  
export class MockData {  
  public static Products: Product[] = [  
    {  
      'id': 11,  
      'modelName': 'F5 Youth',  
      'color': 'Gold',  
      'productType': 'Mobile',  
      'brand': 'OPPO',  
      'price': 16990  
    },  
    {  
      'id': 12,  
      'modelName': 'Inspiron',  
      'color': 'Gray',  
      'productType': 'Laptop',  
      'brand': 'DELL',  
      'price': 59990  
    }  
  ];  
}
```

mock-product-data.ts

Mock Data



ProductService
products : Product[]

ProductsComponent
constructor(productService : ProductService)

AddProductComponent
constructor(productService : ProductService)

OtherComponents
constructor(productService : ProductService)

Lista komentarzy – implementacja usługi

- Przenosimy dane z pliku component-list do pliku mock.ts (symulującego źródło danych).
- Następnie tworzymy serwis udostepniający dane pochodzące z mock

Dlaczego usługa obsługi danych?

- Użytkownik usługi nie wie z jakiego źródła są dane.
- Dane mogą pochodzić z Web Serwisu, z lokalnego pliku albo być imitowane.
- To jest piękno korzystania z usług!
- Usługa odpowiada za dostęp do danych.
- W każdej chwili można zmienić sposób dostępu - zmiany są tylko w tej jednej usłudze.

Lista komentarzy

Realizacja usługi udostępniającej dane

```
import { Comment } from '../comment';  
export const KomentarzeDane: Comment[] = [  
  {imie: "Grzegorz", komentarz: "Pierwszy komentarz", hidden: true },  
  {imie: "Anna", komentarz: "Super strona", hidden: false },  
  {imie: "Alicja", komentarz: "Fajny film wczoraj widziałam", hidden: true },  
];
```

```
import { Injectable } from '@angular/core';  
import { KomentarzeDane } from '../mock';  
@Injectable()  
export class KomentarzeService {  
  getComments() {  
    // return komentarze;  
    return Promise.resolve(KomentarzeDane);  
  }  
}
```

Komunikacja asynchroniczna

```
constructor( service: KomentarzeService ) {  
  
  this.comments = service.getComments();  
}
```