# Building Application using Angular

Dependency Injection

## Dependency Injection

- Code without DI-drawbacks
- DI as a design pattern
- DI as a framework

## Code without DI

```
class Bricks
{
    constructor(parameter)
    {

    }
}
class Cements
{
    constructor()
    {

    }
}
class House
{
    bricks;
    cements;
    constructor()
    {
        this.bricks=new Bricks();
        this.cements=new Cements();
    }
}
```

Drawback code is not flexible. If the dependency has changed House class need to be changed as well .Code is not suitable for testing.

## DI as a design pattern continued..

- DI is a coding pattern in which a class receives its dependencies from external sources rather than creating them itself.

Without DI

```
class House
{
    bricks;
    cements;
    constructor()
    {
        this.bricks=new Bricks();
        this.cements=new Cements();
    }
}
```

With DI

```
class House
{
    bricks;
    cements;
    constructor(bricks,cements)
    {
        this.bricks=bricks;
        this.cements=cements;
    }
}
```

## DI as a design pattern continued..

- Code is much flexible now

```
var brick=new Bricks();
var cement=new Cements();
var house=new House(brick,cement);

var brick=new Bricks(parameter);
var cement=new Cements();
var house=new House(brick,cement);

var brick=new Bricks(parameter);
var cement=new Cements();
var house=new House(brick,cement);
```

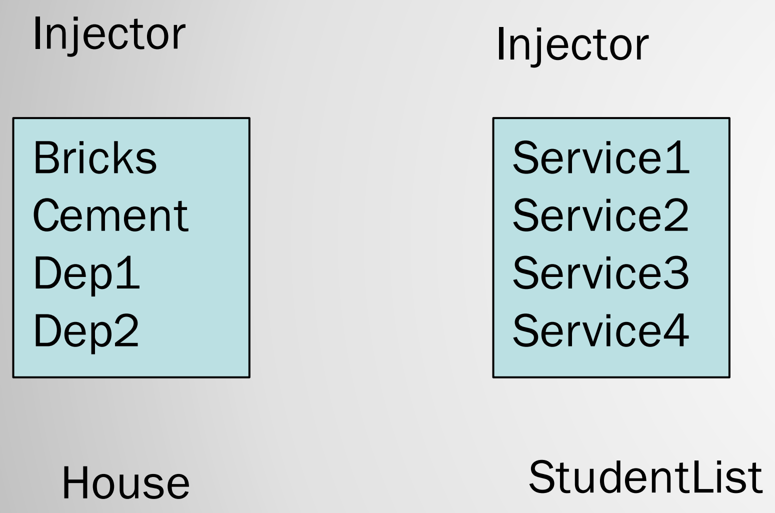
## DI as a design pattern continued..

```
var brick=new Bricks();
var cement=new Cements();
var dep1=new dependency();
var dep2=new dependecy();
var house=new House(brick,cement,dep1,dep2);

var brick=new Bricks();
var cement=new Cements();
var dep1=new dependency();
var dep2=new dependecy(dep1);
var house=new House(brick,cement,dep2);
```

## DI as a Framework

- Injector is basically container for all the dependency



## DI as a framework contd

- Define the StudentService class

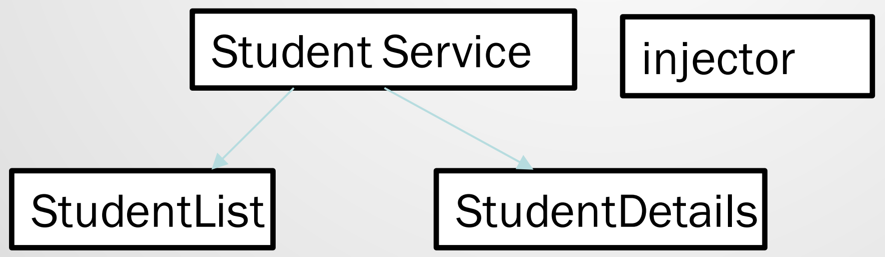
```
ng g s Student
```

- Register with Injector

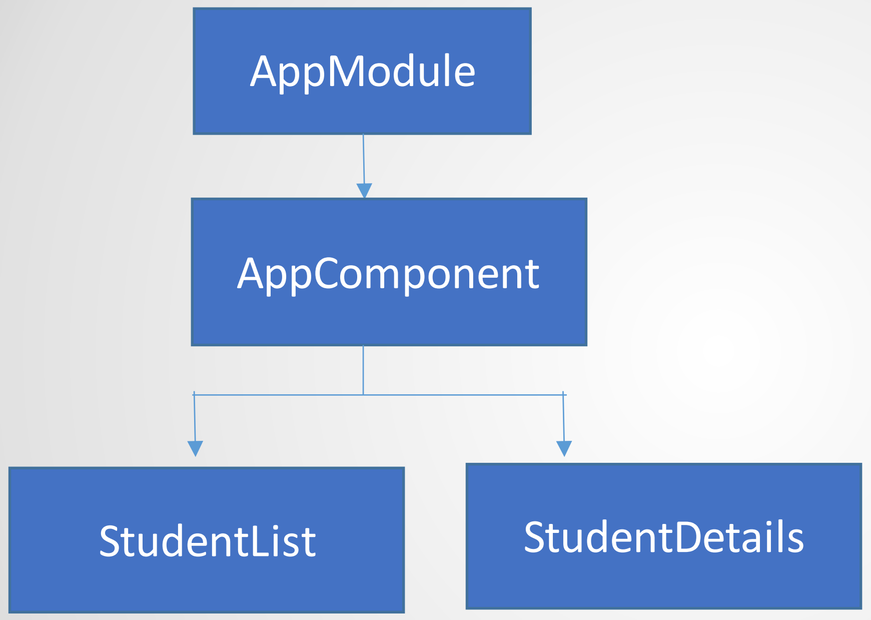
```
providers: [StudentService]
```

- Declare as dependency in StudentList and StudentDetails component

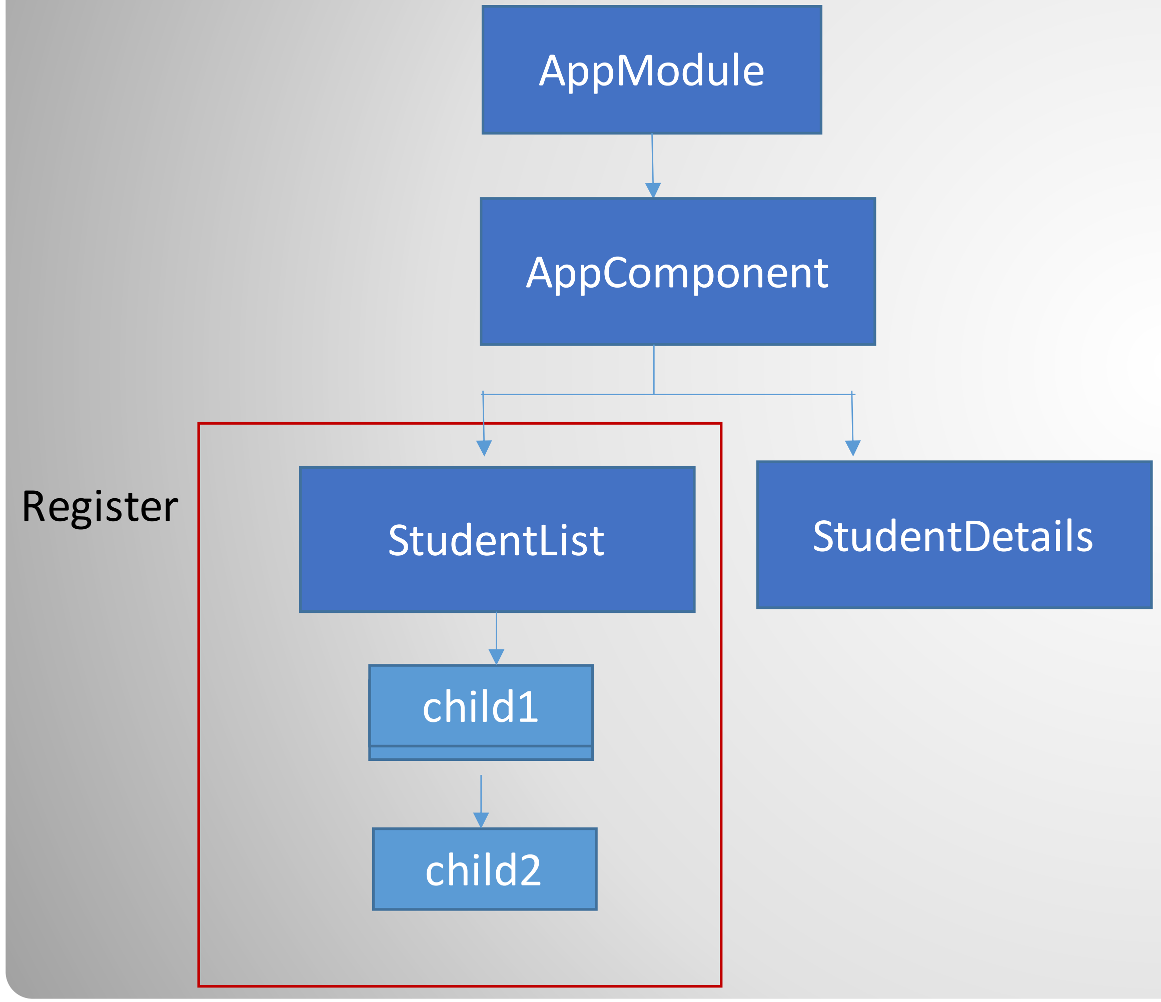
```
constructor(private listservice:StudentService) { }
```



## Hierarchical DI in Angular



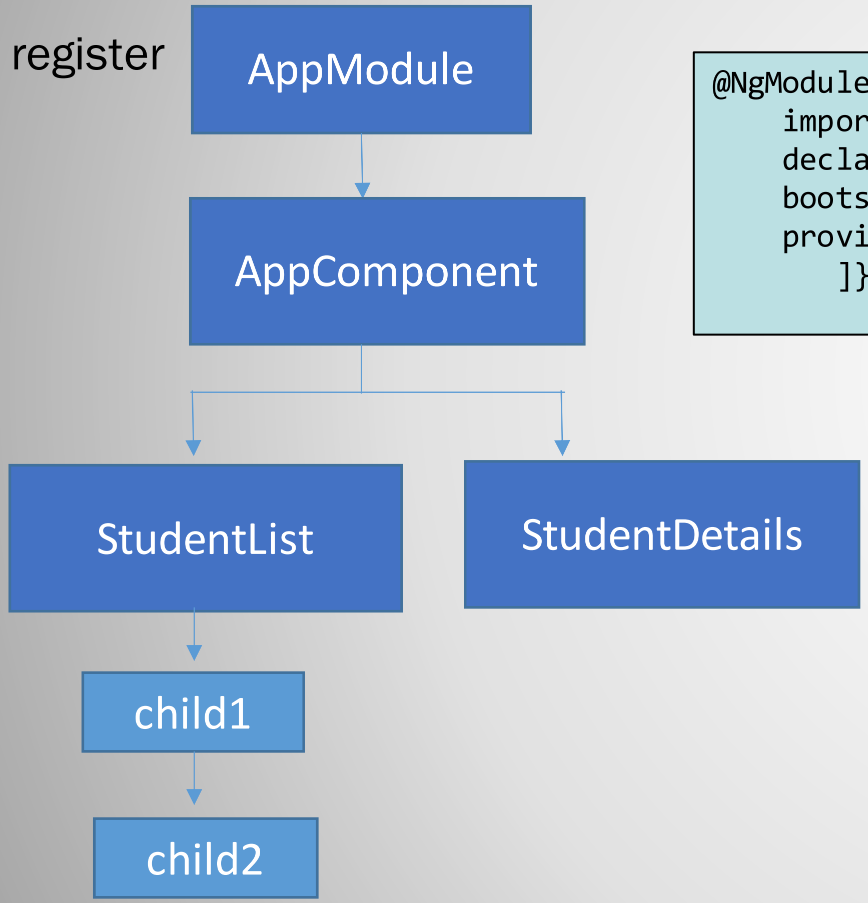
# Hierarchical DI in Angular



Code snippet

```
import { Component, OnInit } from '@angular/core';
import {StudentService} from '../student.service';
@Component({
  selector: 'app-student-list',
  templateUrl: './student-list.component.html',
  providers:[StudentService],
  styleUrls: ['./student-list.component.css']
})
```

## Hierarchical DI in Angular



```
@NgModule({
    imports:      [ BrowserModule, FormsModule ],
    declarations: [ AppComponent,  StudentListComponent, StudentDetailsComponent,
    bootstrap:    [ AppComponent ],
    providers: [StudentService, LoggerService,
        ]})
```

## Service is injected by another Service

- @injectable decorator while creating service or while injecting in constructor We need to define @inject(servicename)

service

```
import { Injectable } from '@angular/core';
 @Injectable
export class LoggerService {
   constructor() { }
  public log(name:string)
  {
console.log("This is "+ name+" method name");
  }
 }
```

Injecting service

```
import { Injectable,Inject } from '@angular/
core';
import { LoggerService } from './logger.serv
ice';
import { Observable } from 'rxjs';
 @Injectable()
export class ListDataService {
list:number[]=[];
  constructor(@Inject(LoggerService)private
loggerServcie) { }
```

## Value Data Service

```
import { Inject } from '@angular/core';
//constant variable as a service
export const API_URL:string="API_URL";

export class ValueDataService {

//injecting constant variable servivce
constructor(@Inject(API_URL) private apiUrl: string) { }
  get(): void {
    console.log(`Calling ${this.apiUrl}/endpoint...`);
  }
}
```

## Registering Value Data Service

```
@NgModule({
    imports:      [ BrowserModule, FormsModule ],
    declarations: [  ValueDataComponent],
    bootstrap:    [ AppComponent ],
    providers: [
     { provide: ValueDataService, useClass: ValueDataService },
      {
        provide: API_URL,
        useValue: 'https://production-api.sample.com',
          }
    ]})
```

## Using observables to pass values

- Observables provide support for passing messages between parts of your application. They are used frequently in Angular and are the recommended technique for event handling, asynchronous programming, and handling multiple values.

## Using observables to pass values

```
export class AppModule { }
import { Observable } from 'rxjs';
  public getnumbers(): any {
      const numbersObservable = new Observable(observer => {
             setTimeout(() => {
                 observer.next(this.list);
             }, 1000);
      });

      return numbersObservable;
    }
```

## Calling Observables

```
import { Component, OnInit } from '@angular/core';
import {StudentService} from '../student.service';
@Component({
  selector: 'app-student-list',
  templateUrl: './student-list.component.html',
  providers:[StudentService],
  styleUrls: ['./student-list.component.css']
})
export class StudentListComponent implements OnInit {
studentList:Array<any>=[];
  constructor(private listservice:StudentService) { }

  ngOnInit() {
          const observablelist = this.listservice.getobservableList();
        observablelist.subscribe((studentlist: any[]) => {
            this.studentList = studentlist;
        });
  }
}
```

## Link

- https://stackblitz.com/edit/dependencydemo
- https://angular.io/guide/dependency-injection