DEPENDENCY INJECTION

TBC

📖 DI

Dependencies are services or objects that a class needs to perform its function. DI is a coding pattern in which a class asks for dependencies from external sources rather than creating them itself.

📖 DI

```
@Injectable({
  // we declare that this service should be created
  // by the root application injector.
  providedIn: 'root',
})
export class HeroService {
  getHeroes() { return HEROES; }
}
```

📖 DI

- @Injectable()
- @NgModule()
- @Component()

🕮 DI

```
export class HeroListComponent {

  heroes: Hero[];



  constructor(heroService: HeroService) {

    this.heroes = heroService.getHeroes();

  }

}
```

📖 DI

```
@Injectable({
  providedIn: 'root',
})
export class HeroService {

  constructor(private logger: Logger) {  }

  getHeroes() {
    this.logger.log('Getting heroes ...');
    return HEROES;
  }
}
```

📖 DI

```
@Injectable({
  providedIn: 'root'
})
export class Logger {
  logs: string[] = []; // capture logs for testing

  log(message: string) {
    this.logs.push(message);
    console.log(message);
  }
}
```

TBC

🕮 DI OPTIONAL

Returns null if not found in DI

```
constructor(@Optional() private logger: Logger) {
  if (this.logger) {
    this.logger.log(some_message);
  }
}
```

🕮 INJECTABLE LEVEL

```
@Injectable({
  // we declare that this service should be created
  // by any injector that includes HeroModule.
  providedIn: HeroModule,
})
export class HeroService {
  getHeroes() { return HEROES; }
}
```

TBC

🕮 NGMODULE LEVEL

```
@Injectable()
export class Service {
  doSomething(): void {
  }
}

@NgModule({
  providers: [Service],
})
export class ServiceModule {
}
```

TBC

🕮 COMPONENT LEVEL

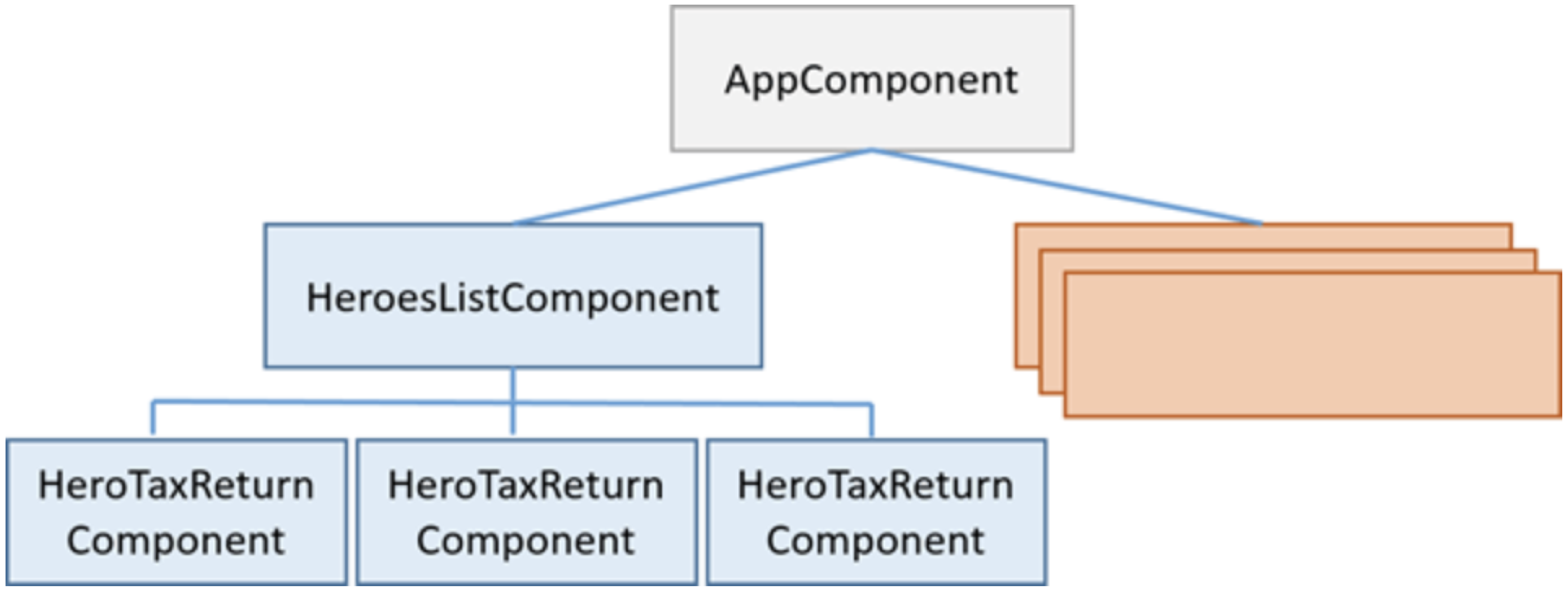
Individual components within an NgModule have their own injectors. You can limit the scope of a provider to a component and its children by configuring the provider at the component level using the @Component metadata.

🕮 COMPONENT LEVEL

```
@Component({
  selector: 'app-heroes',
  providers: [ HeroService ],
  template: `
    <h2>Heroes</h2>
    <app-hero-list></app-hero-list>
  `

})
export class HeroesComponent { }
```

📖 INJECT CHECKING



PROVIDER OBJECT LITERAL

```
@Injectable()
export class EvenBetterLogger extends Logger {
  constructor(private userService: UserService) { super(); }

  log(message: string) {
    let name = this.userService.user.name;
    super.log(`Message to ${name}: ${message}`);
  }
}
```

```
[ UserService,
  { provide: Logger, useClass: EvenBetterLogger }]
```

🕮 PROVIDER OBJECT LITERAL

```
[ NewLogger,
  // Not aliased! Creates two instances of `NewLogger`
  { provide: OldLogger, useClass: NewLogger}]
```

```
[ NewLogger,
  // Alias OldLogger w/ reference to NewLogger
  { provide: OldLogger, useExisting: NewLogger}]
```

TBC

🕮 VALUE PROVIDER

```
// An object in the shape of the logger service
export function SilentLoggerFn() {}

const silentLogger = {
  logs: ['Silent logger says "Shhhhh!". Provided via "useValue"'],
  log: SilentLoggerFn
};
```

```
[{ provide: Logger, useValue: silentLogger }]
```

WITH FACTORY PROVIDER

```
const heroServiceFactory = (logger: Logger, userService: UserService) => {
  return new HeroService(logger, userService.user.isAuthorized);
};
```

```
export let heroServiceProvider =
  { provide: HeroService,
    useFactory: heroServiceFactory,
    deps: [Logger, UserService]
  };
```

```
@Component({
  selector: 'app-heroes',
  providers: [ heroServiceProvider ],
  template: `
    <h2>Heroes</h2>
    <app-hero-list></app-hero-list>
  `
})
export class HeroesComponent { }
```

HERE GOES!

TBC