Understanding Angular Dependency Injection

Jim Cooper
Software Engineer

@jimthecoop | jcoop.io

Q: What is Dependency Injection?

Dependency Injection

```
class ProductComponent {




}
```

```
class ProductRepository {
  get(id: string) { … }
  update(product: Product) { … }
}
```

Dependency Injection

```
class ProductComponent {
  private repo: ProductRepository;

  constructor() {
    this.repo = new ProductRepository();
  }



}
```

```
class ProductRepository {
  get(id: string) { … }

  update(product: Product) { … }
}
```

Dependency Injection

```
class ProductComponent {
  private repo: ProductRepository;

  constructor() {
    this.repo = new ProductRepository();
  }


}
```

```
class ProductRepository {
  constructor(dbConnection) { }
  get(id: string) { … }
  update(product: Product) { … }
}
```

Dependency Injection

```
class ProductComponent {
  private repo: ProductRepository;

  constructor() {
    const dbConn = new DbConnection();
    this.repo = new ProductRepository();
  }


}
```

```
class ProductRepository {
  constructor(dbConnection) { }
  get(id: string) { … }
  update(product: Product) { … }
}
```

Dependency Injection

```
class ProductComponent {
  private repo: ProductRepository;

  constructor() {
    const connStr = 'a;db;connection;string;';
    const dbConn = new DbConnection( connStr);
    this.repo = new ProductRepository();
  }

}
```

```
class ProductRepository {

  constructor(dbConnection) { }

  get(id: string) { … }

  update(product: Product) { … }
}
```

Dependency Injection

```
class ProductComponent {
  private repo: ProductRepository;

  constructor(prodRepo: ProductRepositry) {
    this.repo = prodRepo;
  }


}
```

```
class ProductRepository {
  constructor(dbConnection) { }
  get(id: string) { … }
  update(product: Product) { … }
}
```

Q: Why is Dependency Injection Useful?

Benefits of Dependency Injection

- Testability
- Flexibility
- Modularity
- Scalability
- Reusability
- Reduced Complexity

Dependency Injection Benefits: Testability

```
class ProductRepository {
  dbConnection:DbConnection;

  constructor(dbConnection:DbConnection);
    this.dbConnection =  dbConnection;
  }

  getFilteredProducts(filter: string) {
    let allProducts = this.dbConnection.getAllProducts();

    return allProducts.filter(...);
  }

}
```

Benefits of Dependency Injection

Testability

Flexibility

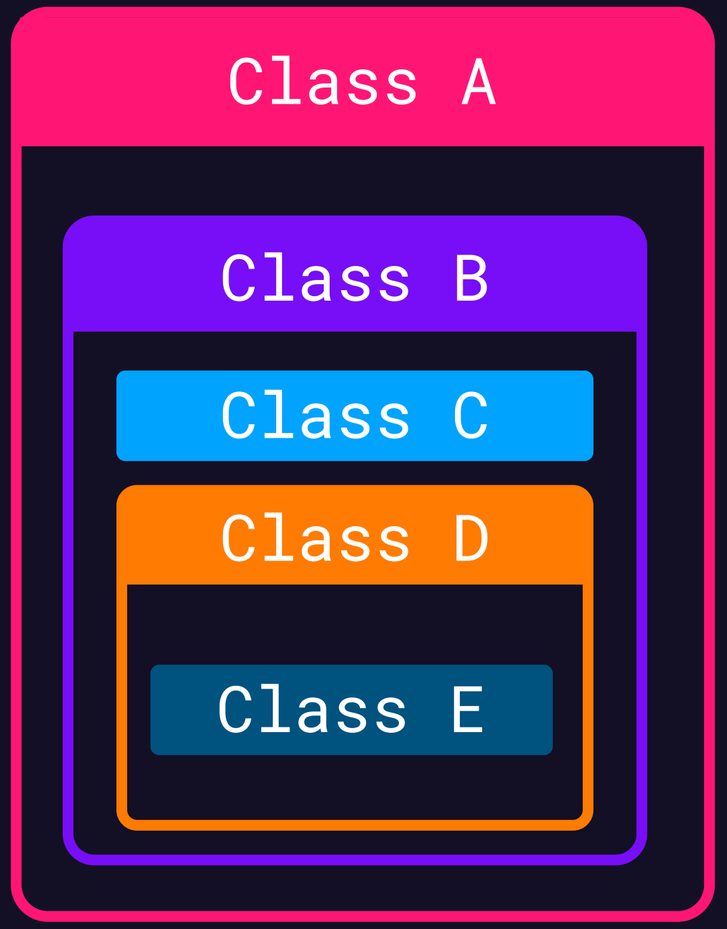
Modularity

Scalability

Reusability

Reduced Complexity

Dependency Injection Benefits: Reduced Complexity




```
let e = new ClassE();
let d = new ClassD(e);
let c = new ClassC();
let b = new ClassB(c, d);
let a = new ClassA(b);
```

Benefits of Dependency Injection

- Testability
- Flexibility
- Modularity
- Scalability
- Reusability
- Reduced Complexity

Dependency Injection Examples

Dependency Injection Example: Service Locator

```
class ServiceLocator {
  private productService: IProductService = new ProductService();
  private dbContext: IDbContext = new RobotStoreDbContext();
  private robotRepository: IRepository<Robot> = new RobotRepository(this.dbContext);

  getProductService(): IProductService {
    return this.productService;
  }

  getDbContext(): IDbContext {
    return this.dbContext;
  }

  getRobotRepository(): IRepository<Robot> {
    return this.robotRepository;
  }
}
```

IoC Containers

IoC = Inversion of Control

IoC Example: InversifyJS

1.

```
class DbContext {...}

class ProductRepository {
  constructor(private dbContext: DbContext) { ...}
}

class ProductComponent {
  constructor(private productRepository: ProductRepository) { ...}
}
```

IoC Example: InversifyJS

1.

```
@Injectable()
class DbContext {...}

@Injectable()
class ProductRepository {
  constructor(private dbContext: DbContext) { ...}
}

@Injectable()
class ProductComponent {
  constructor(private productRepository: ProductRepository) { ...}
}
```

```
@Injectable() class DbContext ...}

@Injectable() class ProductRepository
  constructor(private dbContext: DbContext) { ...}
}

@Injectable() class ProductComponent {
  constructor(private productRepository: ProductRepository) { ...}
}
```

IoC Example: InversifyJS

1.

```
@Injectable() class DbContext ...}

@Injectable() class ProductRepository
  constructor(private dbContext: DbContext) { {..}.}
} }

@Injectable() class ProductComponent {
  constructor(private productRepository: ProductRepository) { {..}.}
} }
```

2.

```
const dbContextSymbol = Symbol.for('DbContext');
const productRepositorySymbol = Symbol.for('ProductRepository');
const productComponentSymbol = Symbol.for('ProductComponent');
```

3.

```
const container = new inversify.Container();
container.bind<DbContext>(dbContextSymbol).to(DbContext);
container.bind<ProductRepository>(productRepositorySymbol).to(ProductRepository);
container.bind<ProductComponent>(productComponentSymbol).to(ProductComponent);
```

4.

```
class ProductRepository {
  constructor(
    @inject(dbContextSymbol) private dbContext: DbContext
  )
}
class ProductComponent {
  constructor(
    @inject(productRepositorySymbol) private productRepository: ProductRepository
  )
}
```

IoC Example: InversifyJS

1.
```
@Injectable() class ProductRepository {
  constructor(private dbContext: DbContext) { ...}
}
```

2.
```
const productRepositorySymbol = Symbol.for('ProductRepository');
```

3.
```
container.bind<ProductRepository>(productRepositorySymbol).to(ProductRepository);
```

4.
```
class ProductComponent {
  constructor(
    @inject(productRepositorySymbol) private productRepository: ProductRepository
  )
}
```

IoC Example: Angular

1.
```
@Injectable() class ProductRepository {
  constructor(private dbContext: DbContext) { ...}
}
```

2.
```
const productRepositorySymbol = Symbol.for('ProductRepository');
```

3.
```
providers: [
  {
    provide: productRepositorySymbol,
    useClass: ProductRepository
  }
]
```

4.
```
class ProductComponent {
  constructor(
    @inject(productRepositorySymbol) private productRepository: ProductRepository
  )
}
```

IoC Example: Angular

1.
```
@Injectable() class ProductRepository {
  constructor(private dbContext: DbContext) { ...}
}
```

2.
```
const productRepositoryToken
  = new InjectionToken<ProductRepository>('ProductRepository');
```

3.
```
providers: [
  {
    provide: productRepositorySymbol,
    useClass: ProductRepository
  }
]
```

4.
```
class ProductComponent {
  constructor(
    @inject(productRepositorySymbol) private productRepository: ProductRepository
  )
}
```

IoC Example: Angular

1.
```
@Injectable() class ProductRepository {
  constructor(private dbContext: DbContext) { ...}
}
```

2.
```
const productRepositoryToken
  = new InjectionToken<ProductRepository>('ProductRepository');
```

3.
```
providers: [
  {
    provide: productRepositorySymbol:
    useClass: ProductRepository
  }
]
```

4.
```
class ProductComponent {
  constructor(private productRepository: ProductRepository) {}
}
```

IoC Example: Angular

1.
```
@Injectable() class ProductRepository {
  constructor(private dbContext: DbContext) { ...}
}
```

2.
```
const productRepositoryToken
  = new InjectionToken<ProductRepository>('ProductRepository');
```

3.
```
providers: [
  {
    provide: productRepositorySymbol:
    useClass: ProductRepository
  }
]
```

4.
```
class ProductComponent {
  constructor(
    @Inject(productRepositorySymbol) private productRepository: ProductRepository
  )
}
```