

[DIP]

! The Dependency Inversion Principle (DIP) states that high level modules should not depend on low level modules; both should depend on abstractions. Abstractions should not depend on details. Details should depend upon abstractions.

DI vs. SL

- Dependency injection (DI)
 - ** Injects dependencies
 - ** Constructor/property injection
- Service Locator (SL)
 - ** Single point of contact
 - ** Static dependency resolver

Demo

- Project [EUMEL Dj](#)
- Mobile app uses service locator
- Desktop app uses dependency injection

Using DI Container

- Reduces "hard dependencies"
- Delegates creation
- Simplifies injecting of code
- Simplifies changing of implementation

👉 A DI container makes you write cleaner software

👍 A DI container helps refactoring code

[Singleton]

- Create once, use the same instance everywhere
- Scope of a singleton can be different
 - ** Static
 - ** Per thread
 - ** Per HTTP request
 - ** Any customer defined
- Implementation depends on requirements
- DI implements singleton with different scopes

[Strategy]

- Inject behaviour
- Strategy changes depending on needs
- DI implementa strategy with registrations
- Some frameworks support "named registrations"

Instance Creation

- `T` aka. Instance
- `Func<T>` aka. Instance Factory
- `Lazy<T>` aka. Lazy Instance

Demo: DI

- Setup (Singleton, Scopes)
- Strategy
- T, Lazy<T>, Func<T>

Container Abstraction

- Container registers itself
- Create "Layer of Abstraction"
- IResolve
 - ** Get<T>
 - ** GetAll<T>