DepIn-O: Validation

[1]Appendix to "DepIn-O: an Ontology on Dependency Injection Software Frameworks"

This appendix presents DepIn-O's validation, performed by mapping DepIn-O's **Catalog of Concepts** to instances for each framework analyzed.

1. Language: Java

1.1. Framework: Spring

Listing 1. Consumer with Constructor Injection Point

```
@Component
public class ConsumerCI {
    DependencyA dep;

    public ConsumerCI(DependencyA da) {      // constructor Injection Point
        dep = da;
    }
}
```

Listing 2. Consumer with Method Injection Point

```
@Component
public class ConsumerMI {
    public void generalMethod(DependencyA da) {}     // method Injection Point
}
```

Listing 3. Consumer with Property Injection Point

```
@Component
public class ConsumerPI{
    @Autowired DependencyA dep;     // property Injection Point
}
```

Listing 4. Abstract Dependency

```
public interface DependencyA {}
```

Listing 5. Primary Dependency

```
@Component @Primary
public class DepImpl implements DependencyA {}
```

Listing 6. Qualified Dependency

```
@Component @Qualifier("Alt")
public class DepImpl2 implements DependencyA {}
```

Listing 7. Dependency Specification (in a Property Injection Point)

```
@Autowired @Qualifier("Alt") DependencyA dep ;
```

Listing 8. Qualifier Key

```
"Alt"
```

Listing 9. Decorator (as Primary Dependency and Consumer of a Qualified Dependency)

```
@Component @Primary
public class DecoratorA implements DependencyA {
  @Autowired @Qualifier("Alt") DependencyA dep ;
}
```

Listing 10. Singleton Dependency

```
@Component @Scope("singleton")
public class DepSing {}
// this is the default scope in Spring
```

Listing 11. Dependent Dependency

```
@Component @Scope("prototype")
public class DepProto {}
```

Listing 12. Request Scoped Dependency

```
@Component @RequestScope
public class DepReq {}
```

Listing 13. Session Scoped Dependency

```
@Component @SessionScope
public class DepSes {}
```

Listing 14. Application Scoped Dependency

```
@Component @ApplicationScope
public class DepApp {}
```

1.2. Framework: CDI

Listing 15. Consumer with Constructor Injection Point

```
public class ConsumerCI {
  DependencyA dep ;

  @Inject
  public ConsumerCI(DependencyA da) {     // constructor Injection Point
    dep = da ;
  }
}
```

Listing 16. Consumer with Method Injection Point

```
public class ConsumerMI{
    public void generalMethod(DependencyA da) {}     // method Injection Point
}
```

Listing 17. Consumer with Property Injection Point

```
public class ConsumerPI {
  @Inject DependencyA dep ;
}
```

Listing 18. Abstract Dependency

```
public interface DependencyA {}
```

Listing 19. Primary Dependency

```
@Default
public class DepImpl implements DependencyA {}

@Alternative
public class DepImpl2 implements DependencyA {}

// The annotation @Default is optional. There must be only one class not annotated as @Alternative
     that will be the Primary Dependency.
```

Listing 20. Qualified Dependency

```
Qualified Dependency
@Named("Impl1")
public class DepImpl implements DependencyA {}

@Named("Impl2")
public class DepImpl2 implements DependencyA {}

// Using the annotation @Named requires all dependencies to be qualified.
```

Listing 21. Dependency Specification (in a Property Injection Point)

```
@Inject @Named("Impl2") DependencyA dep ;
```

Listing 22. Qualifier Key

```
"Impl2"
```

Listing 23. Decorator

```
@Decorator
public class DecoratorA implements DependencyA {
  @Inject @Delegate DependencyA dep ;
}
// delegates dep to be resolved by the dependency implementation that had been previously defined as
     default/primary

// beans.xml
    <decorators>
        <class>{packagepath}.DecoratorA</class>
    </decorators>
```

Listing 24. Singleton Dependency

```
@Singleton
public class DepSing {}
```

Listing 25. Dependent Dependency

```
@Dependent
public class DepDepend {}
// this is the default scope in CDI
```

Listing 26. Request Scoped Dependency

```
@RequestScoped
public class DepReq {}
```

Listing 27. Session Scoped Dependency

```
@SessionScoped
public class DepSes {}
```

Listing 28. Application Scoped Dependency

```
@ApplicationScoped
public class DepApp {}
```

2. Language: Python

2.1. Framework: Dependency Injector

Listing 29. Consumer with Constructor Injector

```
class ConsumerCI:
  def __init__(self, service : DependencyA):
    self.dependency = service
```

Listing 30. Consumer with Method Injector

```
class ConsumerMI:
  def test(self, service : DependencyA):
```

Consumer (with Property Injector): not possible to implement in Python/Dependency Injector.

Listing 31. Abstract Dependency

```
class DependencyA(metaclass=abc.ABCMeta):
```

Listing 32. Dependency Implementations

```
Dependency Implementations
class DepImpl1(DependencyA):

class DepImpl2(DependencyA):
```

Listing 33. Decorator

```
class DecoratorA(DependencyA):      #decorator
  def __init__(self, service):
     self.dep = service
```

Listing 34. Dependency Injection (in the Container)

```
class Container(containers.DeclarativeContainer):
        dep = providers.Factory(DependencyA)
  deco = providers.Factory(DecoratorA, dep)
  con = providers.Factory(ConsumerCI, deco)
```

Listing 35. Dependency Specification (in the Container)

```
container = Container()
container.dep.override(providers.Factory(DepImpl1))
```

Primary Dependency & Qualified Dependency: Since Dependency Specification is always required (either the concrete dependency is directly provided or *Abstract-Factory* must be overridden by providing the concrete dependency), all Dependency Implementations fit our Qualified Dependency definition, and none of them fit our Primary Dependency definition. In this case, the **Qualifier Key** is the class name itself.

Listing 36. Singleton Dependency

```
class Container(containers.DeclarativeContainer):
  depSing = providers.Singleton(DepImpl1)
```

Listing 37. Dependent Dependency

```
class Container(containers.DeclarativeContainer):
  depDependent = providers.Factory(DepImpl1)      #Factory = Dependent scope
```

Listing 38. Request Scoped Dependency

```
#resets Container/Dependencies with every new request

class GetReq(BaseHTTPRequestHandler):
   # code

class WebServer(HTTPServer):
  def service_actions(self):
  # if GetReq signals a new request then
     self.depedency = Container().dep()
```

Session Scoped Dependency: reset Container/Dependencies with every new Session, using cookies. Depends on integration with another framework to be implemented.

Application Scoped Dependency: never reset the Container/Dependencies.

3. Language: C#

3.1. Framework: Autofac

The container builder is imperative to support many of the concepts' instantiation, so we start showing how it is created and prepared for usage first.

Listing 39. Container set up

```
var builder = new ContainerBuilder();
// code
var container = builder.Build();
```

Listing 40. Consumer with Constructor Injector

```
public class ConsumerCI{
    IDependencyA dep ;

    public ConsumerCI(IDependencyA service){
        dep = service ;
    }
}
```

Listing 41. Consumer with Method Injector

```
public class ConsumerMI{
  public void test(IDependencyA dep){}
}
```

Listing 42. Consumer with Property Injector

```
public class ConsumerPI{
    public required IDependencyA dep { get; init; }
}
```

Listing 43. Abstract Dependency

```
public interface DependencyA {}
```

Listing 44. Dependency Implementations

```
public class DepImpl1 : IDependencyA{}

public class DepImpl2 : IDependencyA{}
```

Listing 45. Primary Dependency

```
builder.RegisterType<DepImpl1>().As<IDependencyA>();     // sets up DepImpl1 as the Primary Dependency
builder.RegisterType<ConsumerCI>().PropertiesAutowired();    // automatically sets up the injection of
      the primary dependency into the consumer
```

Listing 46. Qualified Dependency

```
builder.RegisterType<DepImpl2>().Keyed<IDependencyA>("Impl2");
```

Listing 47. Dependency Specification via Constructor Injector

```
public ConsumerCI([KeyFilter("Impl2")] IDependencyA service){    // Key in Constructor Method
  dep = service ;
}

// at the container builder
builder.RegisterType<ConsumerCI>().WithAttributeFiltering();     // specifies to look at the filters
     to know which dependency is to be injected
```

Listing 48. Qualifier Key

```
"Impl2";
```

Listing 49. Decorator

```
public class DecoratorA : IDependencyA{
  IDependencyA dep ;

  public DecoratorA(IDependencyA service){
    dep = service ;
  }
}

// at the container builder
builder.RegisterDecorator<DecoratorA, IDependencyA>();
```

Listing 50. Singleton Dependency

```
builder.RegisterType<DepImpl1>().As<IDependencyA>().SingleInstance()
```

Listing 51. Dependent Dependency

```
builder.RegisterType<DepImpl1>().As<IDependencyA>().InstancePerDependency();
// this is the default scope in Autofac
```

Listing 52. Request Scoped Dependency

```
builder.RegisterType<DepImpl1>().As<IDependencyA>().InstancePerRequest();
```

Session Scoped Dependency: According to Autofac's documentation, "This road is fraught with peril and is totally unsupported[1]."

Application Scoped Dependency: declared like the Singleton scope, but in the root container of the application.

3.2. Framework: Simple Injector

The container is imperative to support many of the concepts' instantiation, so we start showing how it is created; when the first call to *GetInstance*, *GetAllInstances* or *Verify* is made, the container locks itself to prevent any future changes being made by explicit registrations.

Listing 53. Container set up

```
var container = new Container();
// code
container.Verify();      // Verify the container's configuration, also locks it.
```

Listing 54. Consumer with Constructor Injector

```
public class ConsumerCI{
    IDependencyA dep;

    public ConsumerCI(IDependencyA service){    // constructor injector
        dep = service;
    }
}

// at the container
container.Register<ConsumerCI>();   // container registration
```

Listing 55. Consumer with Method Injector

```
public class ConsumerMI{
  public void test(IDependencyA dep){}
}
```

Listing 56. Consumer with Property Injector

```
public class ConsumerPI{
    [Import] public IDependencyA dep { get; set; }
}

// additional set-up necessary for property injection to work
class ImportPropertySelectionBehavior : IPropertySelectionBehavior
{
    public bool SelectProperty(Type implementationType, PropertyInfo prop) =>
        prop.GetCustomAttributes(typeof(ImportAttribute)).Any();
}

// at the container
container.Options.PropertySelectionBehavior = new ImportPropertySelectionBehavior();    // enabling
    property injection
container.Register<ConsumerPI>();   // container registration
```

[1] https://autofac.readthedocs.io/en/latest/faq/instance-per-session.html

Listing 57. Abstract Dependency

```
public interface DependencyA {}
```

Listing 58. Dependency Implementations

```
public class DepImpl1 : IDependencyA{}
public class DepImpl2 : IDependencyA{}

// registering them in the container
container.Collection.Append<IDependencyA, DepImpl1>();
container.Collection.Append<IDependencyA, DepImpl2>();
```

Listing 59. Primary Dependency

```
container.Register<IDependencyA, DepImplDef>();
```

Qualified Dependency: According to Simple Injector's documentation: "Resolving instances by a key (**Qualifier Key**) is a feature that is deliberately left out of Simple Injector, because it invariably leads to a design where the application tends to have numerous dependencies on the DI container itself. To resolve a keyed instance you will likely need to call directly into the Container instance and this leads to the Service Locator anti-pattern[2]."

Listing 60. Decorator

```
public class DecoratorA : IDependencyA{
    IDependencyA dep;

    public DecoratorA(IDependencyA service){
        dep = service;
    }
}

// at the container
container.RegisterDecorator<IDependencyA, DecoratorA>();
```

Listing 61. Singleton Dependency

```
container.Register<IDependencyA, DepImplDef>(Lifestyle.Singleton);
```

Listing 62. Dependent Dependency

```
container.Register<IDependencyA, DepImplDef>(Lifestyle.Transient);
// this is the default scope in Simple Injector
```

Listing 63. Request Scoped Dependency

```
container.Options.DefaultScopedLifestyle = new WebRequestLifestyle();
container.Register<IDependencyA, DepImplDef>(Lifestyle.Scoped);
```

Session Scoped Dependency: According to Single Injector's documentation, "This lifestyle is deliberately left out of Simple Injector (...) instead of using Per HTTP Session lifestyle, you will usually be better of by writing a stateless service that can be registered as singleton and let it communicate with the ASP.NET Session cache to handle cached user-specific data." <https://docs.simpleinjector.org/en/latest/lifetimes.html>

Listing 64. Application Scoped Dependency

```
container.Options.DefaultScopedLifestyle = new WcfOperationLifestyle();
container.Register<IUserRepository, SqlUserRepository>(Lifestyle.Scoped);
```

[2]https://docs.simpleinjector.org/en/latest/howto.html