Bindings - KoDeIn - Kotlin Workshop

Tuesday, 14 November 2017
07:15

Bindings

Factory binding

Eine Funktion, die einen definierten Typ als Argument nimmt und ein Objekt des gebundenen Typs zurückgibt. (zB: (A)->T)

```
val kodein = Kodein {
    bind<Dice>() with factory { sides: Int -> RandomDice(sides)
    }
}
```

Provider binding

Bindet einen Typ zu einer Funktion, welche keine Argumente nimmt und ein Objekt des gebundenen Typ zurückgibt (zB: ()->T)

```
val kodein = Kodein { bind<Dice>() with provider {
RandomDice(6) } }
```

Singleton binding

Bindet einen Typ zu einer Instanz die erst beim ersten Aufruf(lazy) und nur einmal instanziert wird.

```
val kodein = Kodein {
    bind<DataSource>() with singleton {
    SqliteDS.open("path/to/file") }
}
```

Tagged binding

Die Abhängigkeiten können markiert werden, so dass verschiedene Instanzen vom gleichen Typ instanziert werden können

```
val kodein = Kodein {
    bind<Dice>() with factory { sides: Int -> RandomDice(sides)
    }
    bind<Dice>("DnD10") with provider { RandomDice(10) }
    bind<Dice>("DnD20") with singleton { RandomDice(20) }
}
```

Multiton binding

Ein Multiton kann man sich als Singleton-Factory vorstellen. Es versichert, immer das gleiche Objekt zurückzugeben für das gleiche Argument.

```
val kodein = Kodein {
    bind<RandomGenerator>() with multiton { max: Int ->
    SecureRandomGenerator(max) }
}
```

Instance binding

Bindet einen Typ zu einer Instanz welche schon existiert

```
val kodein = Kodein {
    bind<DataSource>() with
    instance(SqliteDataSource.open("path/to/file"))
}
```

Scoped Singletons

```
val kodein = Kodein {
    bind<User>() with scopedSingleton(requestScope) {
    User(it.session) }
}
```

Auto Scoped singletons

Fürs Binding ähnlich wie scoped singleton

Der Unterschied liegt darin, dass beim Retrieval der Scope verantwortlich ist für den Context und muss nicht mitgegeben werden. Das Binding zu einem Autoscoped Singleton ist ähnlich wie ein binding zu einem Provider.

Constant binding

```
val kodein = Kodein {
    constant("maxThread") with 8
    constant("serverURL") with "https://my.server.url"
}
```

Transitive dependencies

Eine Klasse welche durhc transitive Abhängigkeiten instanziert werden kann:

```
class Dice(private val random: Random, private val sides:
Int) { /*...*/ }
```

Hier die Bindings:

```
val kodein = Kodein {
    bind<Dice>() with singleton { Dice(instance(),
    instance("max")) }
    bind<Random>() with provider { SecureRandom() }
    constant("max") with 5
}
```