SubCut 2.0

Honestly Truly Completely Absolutely
Simple Dependency Configuration In Scala

SubCut

- Scala
- Uniquely
- Bound
- Classes
- Under
- Traits

Subcutaneous (adjective): being, living, occurring, or administered under the skin (Merriam-Webster Online)

Why Not...?

- Guice
- Spring
- Cake
- Roll-your-own

Guice

```
class ScalaModule extends AbstractModule {
  override protected def configure() {
    bind(classOf[AService]).to(classOf[SomeService])
  }
}

class MyComponent @Inject()(val service: AService) {
  def callTheService(): String = service.service
}
```

Spring

```
<beans>

  <bean id="dog" class="scalaspring.Dog">
    <constructor-arg value="Fido" />
  </bean>

  <bean id="cat" class="scalaspring.Cat">
    <constructor-arg value="Felix" />
  </bean>

</beans>
```

```
val ctx = new ClassPathXmlApplicationContext("applicationContext.
xml")

// instantiate our dog and cat objects from the application
context
val dog = ctx.getBean("dog").asInstanceOf[Animal]
val cat = ctx.getBean("cat").asInstanceOf[Animal]
```

Cake Pattern

```
trait UserRepositoryComponent {
  def userRepository: UserRepository
  trait UserRepository {
    def findAll: List[User]
  }
}

trait ORMUserRepositoryComponent extends UserRepositoryComponent {
  def userRepository = new ORMUserRepository(sf)
  class ORMUserRepositry(val sf: SessionFactory) extends UserRepository {
    def findAll: List[User] = sf.query(...)
  }
}

trait UserServiceComponent {
  def userService: UserService
  trait UserService {
    def findAll: List[User]
  }
}

trait DefaultUserServiceComponent extends UserServiceComponent {
  this: UserRepositoryComponent =>
  def userService = new DefaultUserService
  class DefaultUserService extends UserService {
    def findAll = userRepository.findAll
  }
}
```

Dependency Injection History

- Life Before Dependency Injection?
- Service Location
- Anti-pattern
- Too Harsh?
- Can we learn

Starting Over - Constructor Params

```
class UserManager(db: Db) {
  def findUser(name: String): User = {
    User(db.query(userName = name))
  }
}

class UserPage(db: Db) {
  val userManager = new UserManager(db: Db)

  def showUserDetails(userName: String): HtmlPage = {
    val user = userManager.findUser(userName)
    formatHtml(user)
  }
}

class UserManagerPage(db: Db, httpConfig: HttpConfig) {
  val server = new Server(httpConfig)
  val userPage = new UserPage(db)
  ...
}
```

Constructor Params Problems

```
class Site(db: Db,
           ws1: LdapWebService,
           ws2: CompanyDb,
           txm: TransactionManager,
           ge: GridEngineConfig) // ...
```

Make a Config Object

```
class UserManager(val bindingModule: BindingModule) {
  // ... use config here
}

class UserPage(val bindingModule: BindingModule) {
  val userManager = new UserManager(bindingModule)
}
```

Implicit Parameters to the Rescue

```
class UserManager(implicit val bindingModule: BindingModule) {
  // ... use config here
}

class UserPage(implicit val bindingModule: BindingModule) {
  val userManager = new UserManager   // no need to pass explicitly
}

// other code

implicit val siteBindings = SiteBindings // configured BindingModule
val ump = new UserManagerPage
ump.doSomething()
```

SubCut!

```
import com.escalatesoft.subcut.inject._

trait UserManager extends RestService with Injectable {
  def doIt(): Unit
}

class UserManagerImpl(greet: String)(implicit val bindingModule: BindingModule)
    extends UserManager with Injectable {
  lazy val userPage = inject[UserPage]
  def doIt() {
    userPage.doSomething()
  }
}

trait UserPage extends Injectable {
  lazy val ldapServerUrl = inject[String]('LdapServer)
}

class UserPageImpl(implicit val bindingModule: BindingModule)
    extends UserPage with Injectable {
  // ...
}
```

SubCut Bootstrap

```
object MainApp extends App with Injectable {
  implicit val bindingModule = SiteBindings
  val um = inject [UserManager]
  um.doIt()
}
```

or

```
object MainApp extends App {
  implicit val siteBindings = SiteBindings
  val um = new UserManagerImpl("You look fabulous today!")
  um.doIt()
}
```

Injecting Styles

```
// direct (non-optional)
class DoHickey(implicit val bindingModule: BindingModule) extends Injectable {
  lazy val thang = inject [Thang]
}

// constructor parameter style (guice)
class AnimalDomain(an: Option[Animal] = injected)
    (implicit val bindingModule: BindingModule) extends Home with Injectable {

  lazy val animal = injectIfMissing [Animal] (an)
}

// call with
implicit val animalBindings = AnimalBindings
val pasture = new AnimalDomain(Some(cow))
val desert = new AnimalDomain
```

Optional Injects (Recommended)

```
// optional inject
class DoStuffOnTheWeb(val siteName: String, val date: Date)
    (implicit val bindingModule: BindingModule) extends Injectable {
  lazy val webSearch = injectOptional [WebSearch] getOrElse {
    new BingSearchService
  }

  lazy val maxPoolSize = injectOptional [Int]('maxThreadPoolSize) getOrElse 15

  lazy val flightLookup = injectOptional [FlightLookup] getOrElse {
    new OrbitzFlightLookup
  }

  lazy val session = injectOptional [Session]('currentUser) getOrElse {
    Session.getCurrent()
  }
}
```

Strong Type Checking/Binding

```
lazy val intList = inject [List[Int]]
lazy val stringList = inject [List[String]]
lazy val intListOfListsFoo = inject [List[List[Int]]] ('foo)
lazy val stringListOfListsFoo = inject [List[List[String]]] ('foo)

bind [List[Int]] toSingle List(1, 2, 3)
bind [List[String]] toSingle List("a", "b", "c")
bind [List[List[Int]]] idBy 'foo toSingle List(List(1, 2, 3))
bind [List[List[String]]] idBy 'foo toSingle List(List("a", "b"))

// Does not compile
bind [List[List[String]]] toSingle List("a", "b", "c")
```

Providing the Implicit

```
trait Animal extends Injectable {
  // ...
}

class Dog(implicit val bindingModule: BindingModule)
    extends Animal with Injectable

trait CatBindings {
  implicit val bindingModule: BindingModule = CatConfig
}

class Cat extends Animal with Injectable with CatConfig
```

That Darned Implicit

```
class Some1(implicit val bindingModule: BindingModule)
    extends Injectable
class Some2(param1: String, param2: int)
    (implicit val bindingModule: BindingModule) extends Injectable
class Some3(implicit val bindingModule: BindingModule)
    extends Injectable


// Compiler Plugin!

// in build.sbt
addCompilerPlugin("com.escalatesoft.subcut" %% "subcut" % "2.0")
  // (or 2.0-SNAPSHOT right now)

class Some1 extends AutoInjectable
class Some2(param1: String, param2: String) extends AutoInjectable
class Some3 extends AutoInjectable
```

Bindings - Anatomy

```
object RovingActionModule extends NewBindingModule(module => {

  module.bind [RobotAction] to newInstanceOf [RovingAction]

  module.bind [String] idBy ServerUrl toSingle
    "http://www.escalatesoft.com"

})

// for the ServerUrl ID - New for 2.0!
object ServerUrl extends BindingId
```

Bindings - Singleton Style

```
module.bind [Animal] toSingle { new Cat }

module.bind [Animal] idBy 'Donkey toSingle new Donkey(module)

module.bind [Animal] toModuleSingle {
  module => new Dog("woof")(module)
}

module.bind [Animal] idBy 'LoudDog toModuleSingle {
  implicit module => new Dog("WOOF")
}
```

Bindings - Reflection Style

```
module.bind [Animal] to newInstanceOf [Cat]

module.bind [Animal] idBy 'Donkey to moduleInstanceOf [Donkey]
```

Bindings - Provider Style

```
module.bind [Animal] toProvider { new Cat() }

module.bind [Animal] idBy 'LoudDog toProvider {
  implicit module => new Dog("WOOF")
}
```

Bindings - Unbinding

```
module.bind [Animal] to None   // recommended

module.unbind [Animal]         // can mess up ; inference
```

Bindings - Defining Modules

```
object MainConfig extends NewBindingModule( module => {
  module.bind [Action] toSingle new Walkies
})

implicit val theModule = MainConfig


class ClassConfig extends NewBindingModule( module => {
  module.bind [Action] toSingle new Walkies
})

implicit val aModule = new ClassConfig


import NewBindingModule.newBindingModule

implicit val inlineModule = newBindingModule { module =>
  module.bind [Action] toSingle new Walkies
  module.bind [String] idBy 'Noise toSingle "Woof"
}
```

Importing the Module

```
object MainConfig extends NewBindingModule( module => {
  import module._
  bind [Action] toSingle new Walkies
  bind [String] idBy 'Noise toSingle "Woof"
})

implicit val inlineModule = newBindingModule { module =>
  import module._
  bind [Action] toSingle new Walkies
  bind [String] idBy 'Noise toSingle "Woof"
}
```

Bindings - Identifier

```
object Mammal extends BindingId

lazy val mammal = inject[String](Mammal)
lazy val role = inject[String]('Role)
lazy val vehicle = inject[String]("Vehicle")

implicit val bm = newBindingModule { module =>
  import module._
  bind [String] idBy Mammal toSingle "Vole"
  bind [String] identifiedBy 'Role toSingle "Ride"
  bind [String] idBy "Vehicle" toSingle "Motorbike"
}
```

Module Merging

```
implicit val bm = newBindingModule { module =>
  module <~ RobotProviderModule
  module <~ RovingActionModule                  // will override
  module.bind [Animal] toSingle new Dog         // also overrides
}

implicit val bm = newBindingModule { module =>
  module.mergeWithReplace(RobotProviderModule)
  module.mergeWithReplace(RovingActionModule) // will override
  module.bind [Animal] toSingle new Dog         // also overrides
}

implicit val bm = newBindingModule { module =>
  module.withBindingModules(RovingActionModule,RobotProviderModule)
  // last in wins...
}

implicit val bm = RovingActionModule ~ RobotProviderModule
implicit val bm = RovingActionModule andThen RobotProviderModule
```

Testing with SubCut

```
SomeBindings.modifyBindings { implicit testBindings =>
    // testBindings now holds mutable copy of SomeBindings
  val mock1 = mock[Trait1]
  mock1.greet("fred").returning("hello fred") // the script
  mock1.replay()

  testBindings.bind [Trait1] toSingle mock1  // mock out Trait1

  val greeter = new Greeter          // uses the rebound config
  assert (greeter.greet("fred") === "hello fred")

  mock1.verify()
}
```

Integration with Other Libraries

```
// fails because no implicit module available from container
val page = new InjectablePage

// explicitly bind in standard bindings
class BoundPage extends InjectablePage(SomeBindingModule)

// now works - everything satisfied.
val page = new BoundPage
```

Real World Example

```
describe ("The LocusOfInterestPostProcessor") {
  it("should return a successfully updated LocusOfInterest when all coordinates match"){
    CurationModule.modifyBindings { implicit curationModule =>
      // create a mock coordinate mapper that does what we want
      val mapper = mock[MappingService]
      val varLookup = mock[VariationLookup]
      curationModule.bind[MappingService] toSingle mapper
      curationModule.bind[VariationLookup] toSingle varLookup

      mapper expects 'MapNMtoGenome withArgs ("NM_000055.2:c.1518-8") returning (
        new MappingResult("NM_000055.2:c.1518-8", "16", false, 1701, 1702))
      ensemblVariationLookup expects 'lookUpByRsId withArgs ("rs28933389") returning (
        Some(ChromosomalPositionDetails("16", "1701-1702", 1701, 1702, '+')))

      val loip = new LocusOfInterestPostProcessor
      val processed = loip.postProcess(new LocusOfInterest(1, "GRCh37", "16",
          None, Some("rs28933389"), "CVID0000164",
          Some("NG_009031.1:g.56147"), Some("NM_000055.2:c.1518-8"), None,
          Some("Alias"), ChromosomalPosition.Unchecked ))

      // check that the processed loi has the right values in it
```

About Escalate Software

- Training
  - Open
  - In House / Custom
- Consulting
- Mentoring
- www.escalatesoft.com