



# IoC Containers in Java

**Wolf Schlegel, Valtech UK**

**Eoin Woods, UBS Investment Bank**

# Disclaimer

- This workshop and the materials provided to participants make use of software which constitutes Open-Source software (OSS).
- This software remains the copyright of the original author and you do not, nor is it intended that you will, acquire any right, title or interest in it by virtue of you having received a copy of it from Valtech Limited (“Valtech”).
- The use, redistribution and/or modification of such software is governed by the terms of the appropriate software licence. Workshop participants may use, redistribute and/or modify the software only under the terms of the relevant licence. We emphasise that you should ensure you are fully aware of all licence terms or other terms of use relating to any software that you use. The most common OSS licence terms can be viewed at: http://www.opensource.org/licenses/index.php
- The software is provided as downloaded and is distributed in the hope that it will be useful. However, it is unsupported and is provided "as is" without any warranty or representation of any kind either express or implied including, but not limited to, the implied warranties of merchantability, quality, fitness for a particular purpose , title and non-infringement.
- In no event shall Valtech be liable, whether in contract or tort (including negligence) for any loss or damage arising out of or in connection with the use or performance of Open Source software howsoever obtained. In particular Valtech shall not be liable for any loss of profit, lost savings, business interruption, loss of use, loss of data or loss of business opportunity or any special, indirect, or consequential damages.
- Valtech Limited
  29 January 2006

# Table of Contents


# IoC Containers in Java – Part 1

## Introduction to Inversion of Control

## Discussion of selected IoC Containers

# A short history of IoC

- **IoC frameworks entered the Java world in 2003**
- **Spring 0.9**
  - Open source in February 2003
  - Spring 0.9 released in July 2003
  - Spring 1.0 final released in March 2004
- **PicoContainer**
  - PicoContainer 1.0 beta 1 released in August 2003
- **HiveMind**
  - HiveMind 1.0-rc-1 released in August 2004
- **Microcontainer**
  - Microcontainer 1.0.0 released in September 2005

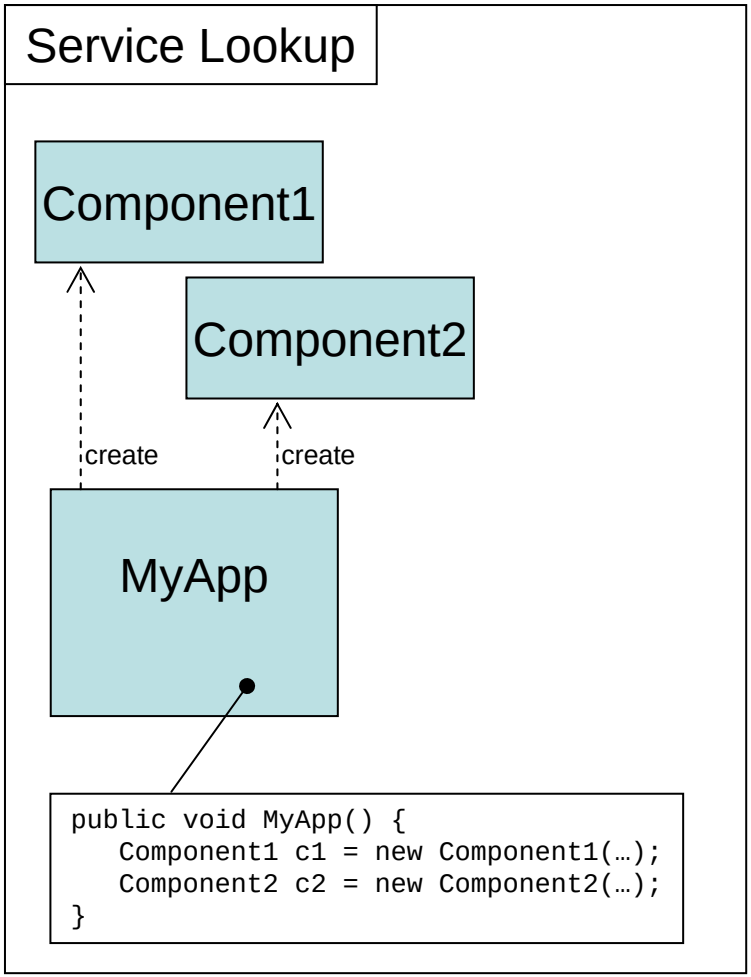
# What is inversion of control?

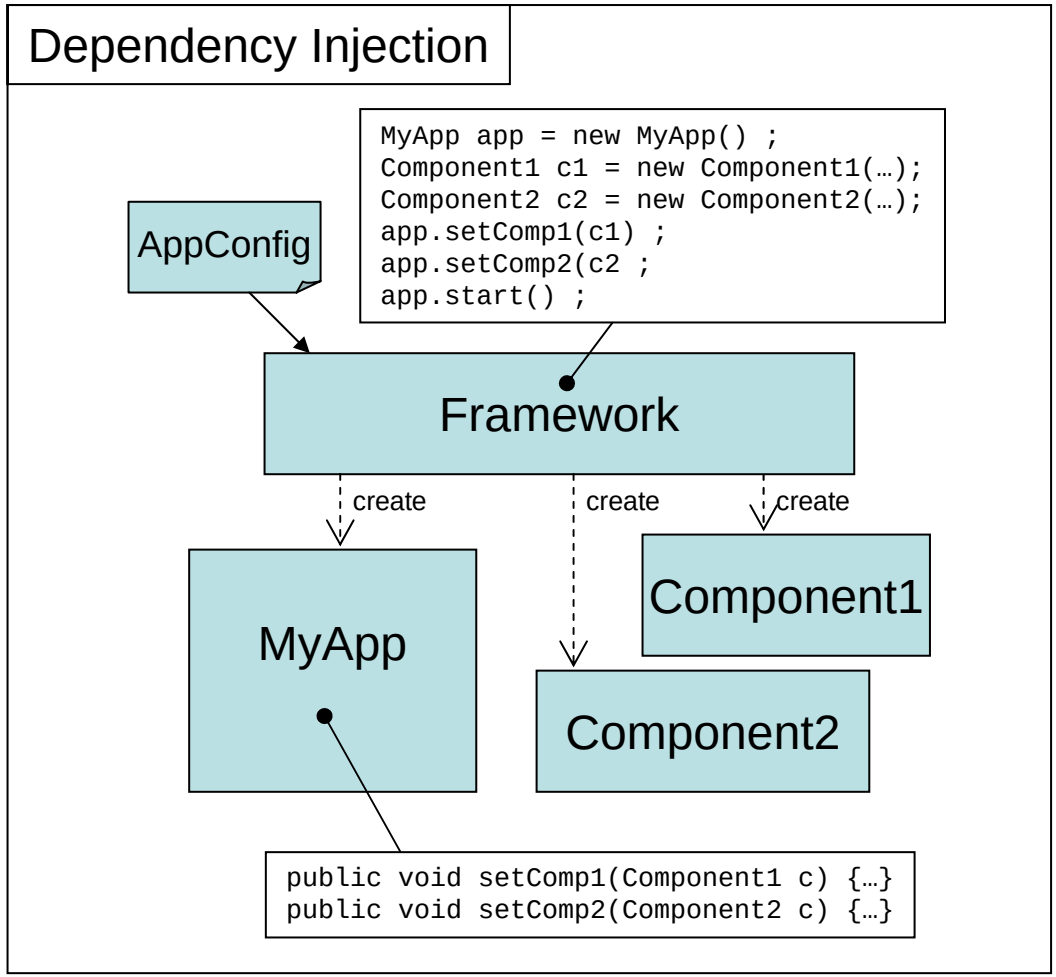
- **General principle exhibited by many frameworks**
- **Traditional software is developed with the application code "in control"**
  - The application defines the "main" function/method/entry point
  - Calls the application's components to perform processing
- **Frameworks invert this relationship and call the application code**
  - "Don't call us we'll call you"
  - Framework provides the "main" function/method/entry point
  - Application code fits into the framework and is called when the framework decides is correct
- **A particular variant of IoC is "dependency injection"**

# What is dependency injection?

- **Most frameworks use the "*Service Lookup*" approach to allow application code to find its dependencies**
  - e.g. J2EE code looks up resources via JNDI
  - CORBA applications find services via a Naming Service
  - The lookup mechanism is hard coded into the application code
- **In contrast, *Dependency Injection* frameworks supply the resources that a component needs when initialising it**
  - Component declares resources required (usually as interface types)
  - Framework configuration defines concrete instances to use
  - Framework passes component the concrete resources at load time
- **Note that most "IoC" containers are really "Dependency Injection" containers**

# Service Lookup vs Dependency Injection



# Why Dependency Injection?

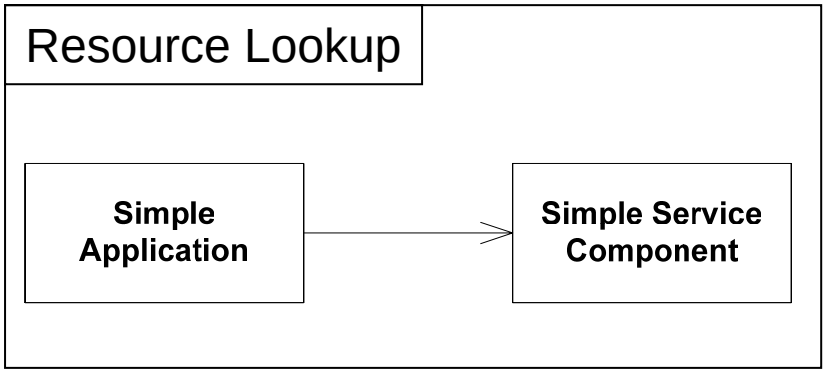
- **Remove dependency on concrete implementations**
  - Allows easy substitution of implementations
  - Increases testability when mocks/stubs can be substituted for components and environment objects (msg queues, db connections, …)
- **Enforced loose coupling**
  - Components have to declare their dependencies via their interface
  - Components must only depend on references passed to them
  - Removes lookup coupling and so makes evolution easier
- **Allow simple component types**
  - Allows frameworks to add standard processing using proxies etc.
  - Hence frameworks can use plain objects as components

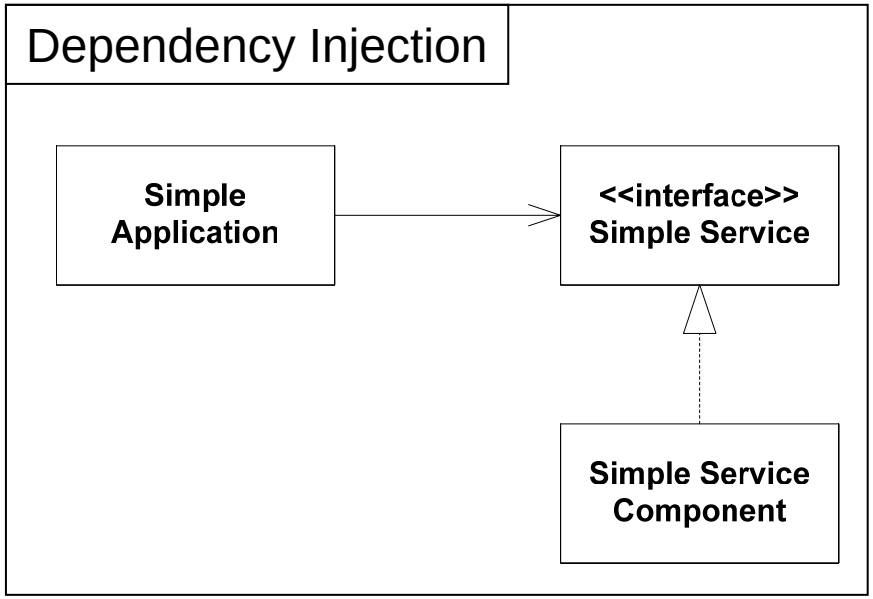
# Designing with Dependency Injection

- **Define dependencies in terms of interfaces (not classes)**
  - Factories aren't required as the IoC Container will do this for us
- **Add setters for each of a class' external dependencies**
  - Or a constructor, most containers allow both approaches
- **Do not look up resources from within a class**
  - If a new resource is required, this is a new dependency
- **Define how the application is assembled using the container**
  - XML configuration file or short code snippet
- **Add an application entry point to initialise the application**
  - Some containers (Spring) need this others (JBoss Microcontainer) don't
  - Initialises the framework, passing it the configuration

# Designing with Dependency Injection

- **Structure and dependencies change slightly …**



# Dependency Injection Containers

- **You can use dependency injection in your application directly**
  - Write a very simple container or hardcode configuration
- **Alternatively, use a container**
  - Provides conventions and configuration
  - Provides factories for components and application assembly code
- **Well known IoC containers**
  - Spring
  - JBoss Microcontainer
  - HiveMind
  - PicoContainer (and NanoContainer)
  - Apache Fortress (in Excalibur)

  Good list at http://java-source.net/open-source/containers

  We'll be looking at Spring, JBoss Microcontainer, HiveMind and PicoContainer

# Implementing Dependency Injection

- **Dependency Injection implemented by many frameworks**
- **Three main approaches taken to its implementation**
  - Constructor, Setter and Interface injection
- **Constructor injection**
  - Declare constructor parameters to define object dependencies
- **Setter injection**
  - Declare JavaBean properties with setters to provide dependencies
- **Interface injection**
  - Implement interfaces that allow container to inject dependencies
- **All approaches require component configuration**
  - Configuration files (e.g. XML) or code

# Implementing Dependency Injection

- **Constructor injection**
  - All dependencies injected via constructor arguments
    - Requires potentially complex constructor argument lists
  - Avoids reliance on get/set naming conventions
    - e.g. `addStateObserver()`
  - Ensures that object is fully initialised after constructor called
    - No need for invariant checking methods
  - Allows immutable properties
    - No need to provide a setter
  - Less flexible initialisation
    - Need a constructor per configuration option
  - Can be difficult to understand
    - Rely on ordering of parameters

# Implementing Dependency Injection

- **Setter Injection**
  - All dependencies injected via setter methods
    - No need for constructors
  - Allows flexible initialisation
    - Any set of properties can be initialised
  - Named properties allow for readable configuration data
    - Clear what is being injected
  - Requires JavaBean conventions to be followed
    - Non standard method names are problematic
  - Can result in invalid object configurations
    - Need post initialisation checking methods
    - Spring `InitializingBean.afterPropertiesSet()` mechanism

# Implementing Dependency Injection

- **Interface Injection**
  - Less common option (Apache Avalon, EJB)
  - Dependencies declared and/or injected via implementing interfaces
  - May couple application to the framework
    - e.g. `javax.ejb.SessionBean.setSessionContext()`
    - Avalon avoids this via configuration
  - Significant implementation overhead for complex dependencies
    - Large number of interfaces to implement and maintain
  - Similar strengths and weaknesses to setter injection

# Implementing Dependency Injection

- **Component Configuration**
  - All three implementation approaches require configuration
  - Configuration may be via configuration files or code
  - Configuration files
    - Usually XML
    - Often awkward to create
    - No type checking (mistakes found at runtime)
  - Code
    - Usually the language of the application code
    - Configuration is “just” coding
    - Type checked via compiler (mistakes found at compile time)

# JBoss Microcontainer (1)

- **“Provides an environment to configure and manage POJOs”**
  - [Getting Started with the JBoss Microcontainer]
- **Comprises a turnkey ready application entry point**
  - org.jboss.kernel.plugins.bootstrap.standalone.StandaloneBootstrap
  - Application code does not need to reference Microcontainer classes
- **Dependencies between POJOs are declared in a configuration file**
  - META-INF\jboss-beans.xml
- **Applications are deployed as .beans files**
  - E.g. calculator.beans
  - Contains the configuration file
  - Applications can be deployed standalone
  - Alternatively, .beans files can be run inside in the JBoss application server

# JBoss Microcontainer (2)

- **Container manages the lifecycle of objects**
- **Configuration of POJOs via jboss-beans.xml comprises**
  - Deploying POJOs known as “beans” using default or other constructors
  - Using factories to create beans
  - Setting properties
  - Injecting dependencies between beans using
    - Constructors
    - Setters
  - Resolving circular dependencies between beans (to a certain extent)
  - Declaring collections containing given elements

# Spring IoC (1)

- **Not just an IoC container!**
  - Extensive enterprise Java application framework
  - MVC webapp framework
  - J2EE / library usability wrappers (e.g. `JmsTemplate`)
- **Simple POJO JavaBeans are combined into applications**
- **Applications constructed using a “Bean Factory”**
  - Number of implementations of …`beans.factory.BeanFactory`
  - `org.springframework.beans.factory.xml.XmlBeanFactory`
- **Provides both setter and constructor injection models**
  - Setter injection often assumed by Spring documentation
  - Bean creation via factories also available for complex situations
- **Beans can be created using constructors or factories**

# Spring IoC (2)

- **Collections of type `List`, `Set`, `Map` and `Properties`**
- **Beans can be created as singletons or non-singletons**
  - The latter is also called a *prototype*
  - The container cannot fully manage the lifecycle of a non-singleton instance
- **Auto-wiring provided by Bean Factories reduce configuration required**
  - Autowiring comes in different modes (e.g. by name or by type)
  - Autowiring by type relies on uniqueness of types used
  - Stubs and implemented beans collide when using autowiring by type
- **Usually possible to avoid any application code dependency on Spring classes and interfaces**

# Spring IoC (3)

- **Lifecycle management via standard interface methods**
  - `InitializingBean.afterPropertiesSet`
  - `DisposableBean.destroy()`
- **Method injection replaces methods in a managed bean**
  - Can be used to overcome incompatible lifecycles of different beans
- **Configuration typically via spring-beans.xml file**
  - Using the XmlBeanFactory
  - Code based configuration is also possible
- **Deployment just involves creating a JAR containing classes and (usually) an XML configuration file**
  - Create bean factory and retrieve main bean to start application

# PicoContainer

- **PicoContainer manages POJOs**
  - Non-intrusive as POJOs need not to depend on PicoContainer classes
- **Supports constructor and setter injection**
- **Lifecycle management based on miscellaneous interfaces**
- **Can monitor lifecycle of components**
- **Supports injection of collections**
- **“No forced metadata choice”**
  - However, NanoContainer supports XML configuration files
- **Nested containers or container hierarchy**
  - Parent is used for resolving components not found locally
  - Components in child-containers can override components in parent-containers

# NanoContainer

- **Provides additional functionality to PicoContainer**
  - In particular meta-data and script language support: XML, Groovy, Beanshell, Jython, Rhino (Javascript)
  - Class name based composition via reflection
  - Classloader management
  - Booter and Deployer
- **NanoContainer's core component is complemented by further components for**
  - Persistence
  - Remoting
  - Adaptors to other IoC frameworks

# HiveMind (1)

- **A sophisticated IoC container**
  - More than just IoC, less than an application framework like Spring
- **Hosted as part of Apache's Jakarta project**
  - http://jakarta.apache.org/hivemind
- **Written by WebCT Inc for their Vista product (2001/02/03)**
  - Then donated to Apache, development still led by Howard Lewis Ship
- **Aimed specifically at J2EE projects**
  - Not a requirement, a general assumption (still usable outside container)
  - Also works well with Jakarta Tapestry webapp framework
- **Supplied in three parts**
  - HiveMind: the IoC container, interceptors and application configuration
  - HiveMind-Lib: Spring and EJB interoperability, configuration helpers
  - HiveMind-JMX: easy service management via JMX

# HiveMind (2)

- **Applications defined as “modules” of “services”**
  - A service is a POJO providing an application component
- **Service construction via a BuilderFactory creating a Registry**
  - Defined via a module descriptor
  - XML or Groovy descriptor files
  - Lazy creation when services are retrieved from the registry
- **Sophisticated dependency injection features**
  - Configuration points for XML to Java object conversion
  - Lightweight initialisation for configuration properties
- **Can support property or factory service creation**
  - Factory creation allows constructor injection or difficult cases
- **Provides unique HiveDoc tool**
  - Generates HiveMind specific application documentation a la JavaDoc

# IoC and Testing

- **IoC allows separation of interface and implementation**
  - Fundamental requirement for easy testing
- **This separation allows easy introduction of mocks or stubs**
  - A “stub” is a simple testing implementation of an interface returning standard values
  - A “mock” is a more sophisticated version that is initialised with an “expectation” (Fowler) of the calls that will be made to it
- **IoC (or rather DI) allows mock or stub objects to be injected**
  - A unit test can inject mock or stub objects
  - A system or integration test can inject simplified implementations
  - Production deployment injects real (complex) service implementations

# Limitations of Dependency Injection

- **Imposes application constraints**
  - Application structure must fit container constraints (e.g. JavaBeans)
  - Naming may need to follow conventions (setters, interfaces, …)
- **Configuration may be awkward**
  - XML files to understand and create
  - Something else for developers to learn
- **Some containers require dependencies on the container**
  - Spring (in many cases), Avalon, EJB, …
- **Edge cases can be very awkward to resolve**
  - Complex dependencies (e.g. maps or lists of dependencies)
  - May need container specific solutions like custom factories
- **Multithreaded applications require a different bean lifecycle**

# Other Runtime Containers

- **Services or infrastructure provided by J2EE containers**
  - Concurrency
  - Messaging
  - Naming and directory
  - Persistence
  - Remoting
  - Transaction management
- **IoC containers "just" assemble an application from POJOs**
  - Generally, they do not address the concerns listed above
    - Spring and NanoContainer provide respective adapters and frameworks
  - However, IoC containers can be run inside J2EE containers which in turn provide these services
    - JBoss MC ".beans" applications can be run inside JBoss Application Server

# Conclusion

- **IoC Containers and Frameworks have much to offer**
  - Demand loose (implementation) coupling of components
  - Unit testing becomes simple, therefore it gets done
  - Frameworks (e.g. Spring) provide a great deal of useful utility code
- **Some resulting technical issues to be aware of**
  - Systems are harder to assemble, comprehend and debug
    - "What **is** an IWidget? Well, it depends …."
  - Only one sort of coupling is addressed
- **Some potential adoption pitfalls to be aware of**
  - Overheads of learning and making mistakes
  - Allow for open source style documentation and supporting materials
  - Decide whether to tie to a container or not: some are quite intrusive

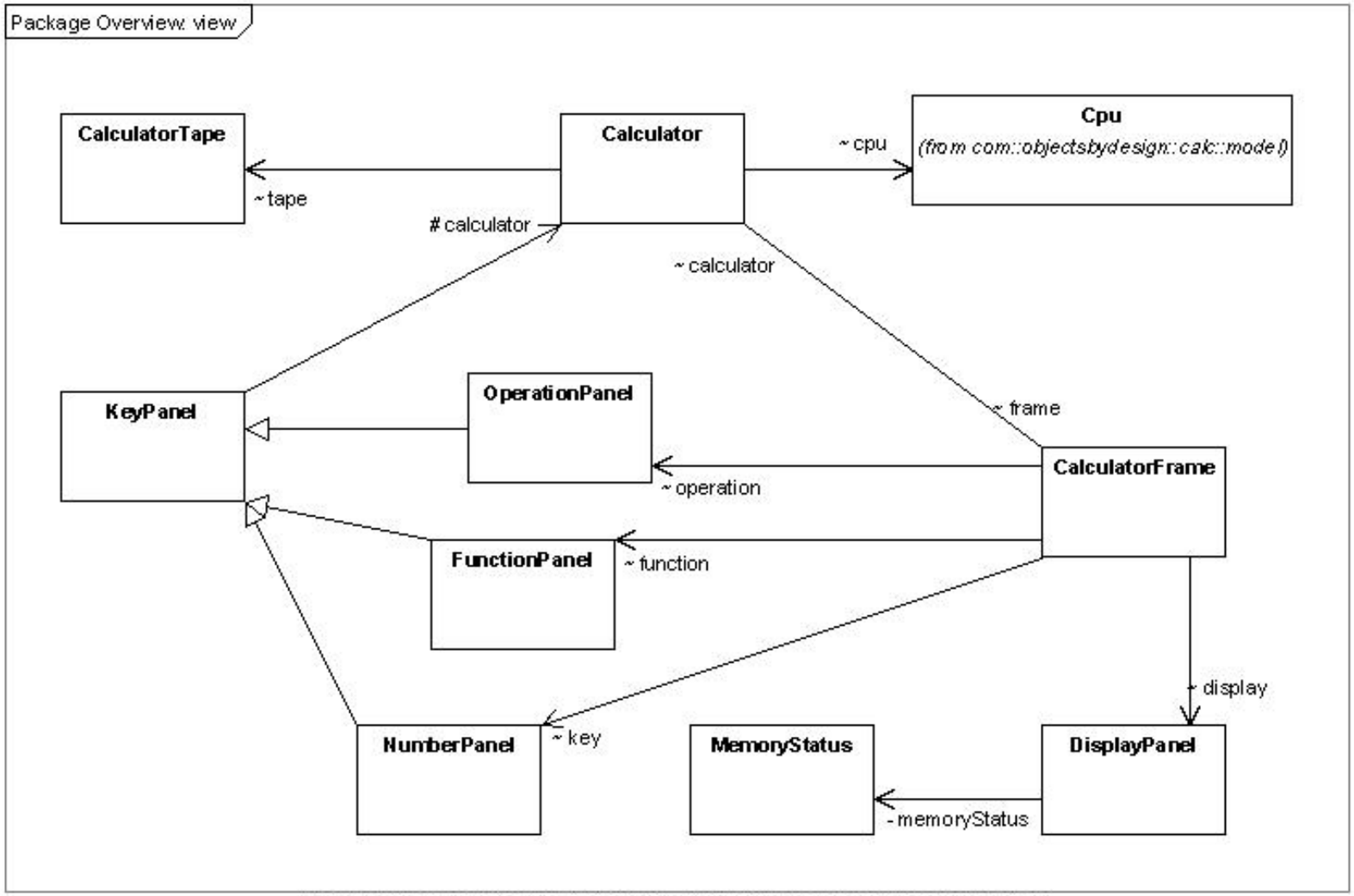

# IoC Containers in Java – Part 2

## Hands- on Exercises

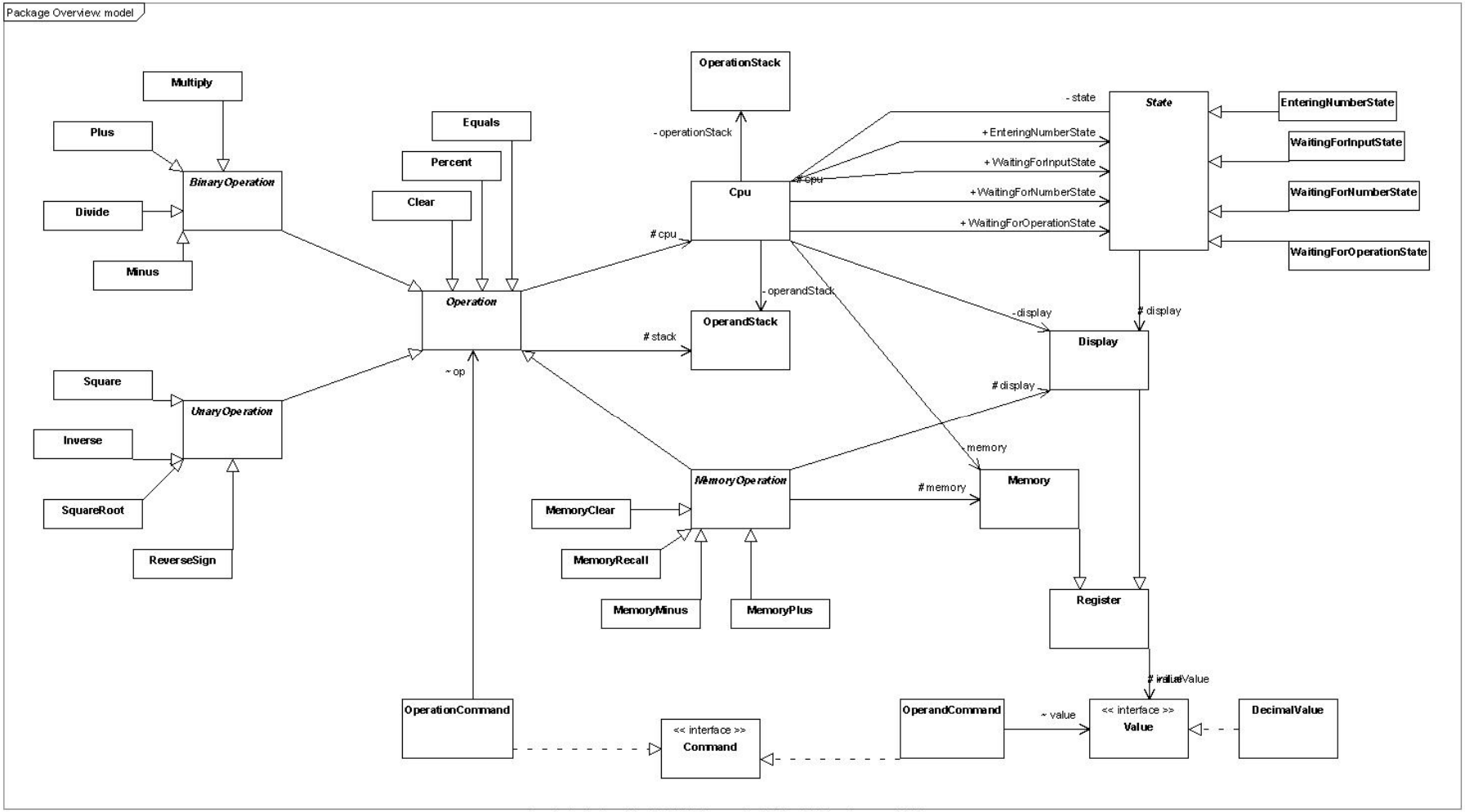
# Calculator Sample Application

- **Calculator provided by Objects by Design**
  - http://www.objectsbydesign.com/projects/calc/overview.html
- **Calculator consists of a couple of components such as**
  - CPU
  - CalculatorTape
  - CalculatorFrame
- **Swing based user interface**
- **A lot of examples provided by IoC containers are**
  - Web applications or
  - Simple classes that illustrate a given detail of the respective container
- **Compared to this, the calculator application is a meaningful and self contained example that can be run from the command line**

# Calculator View Package

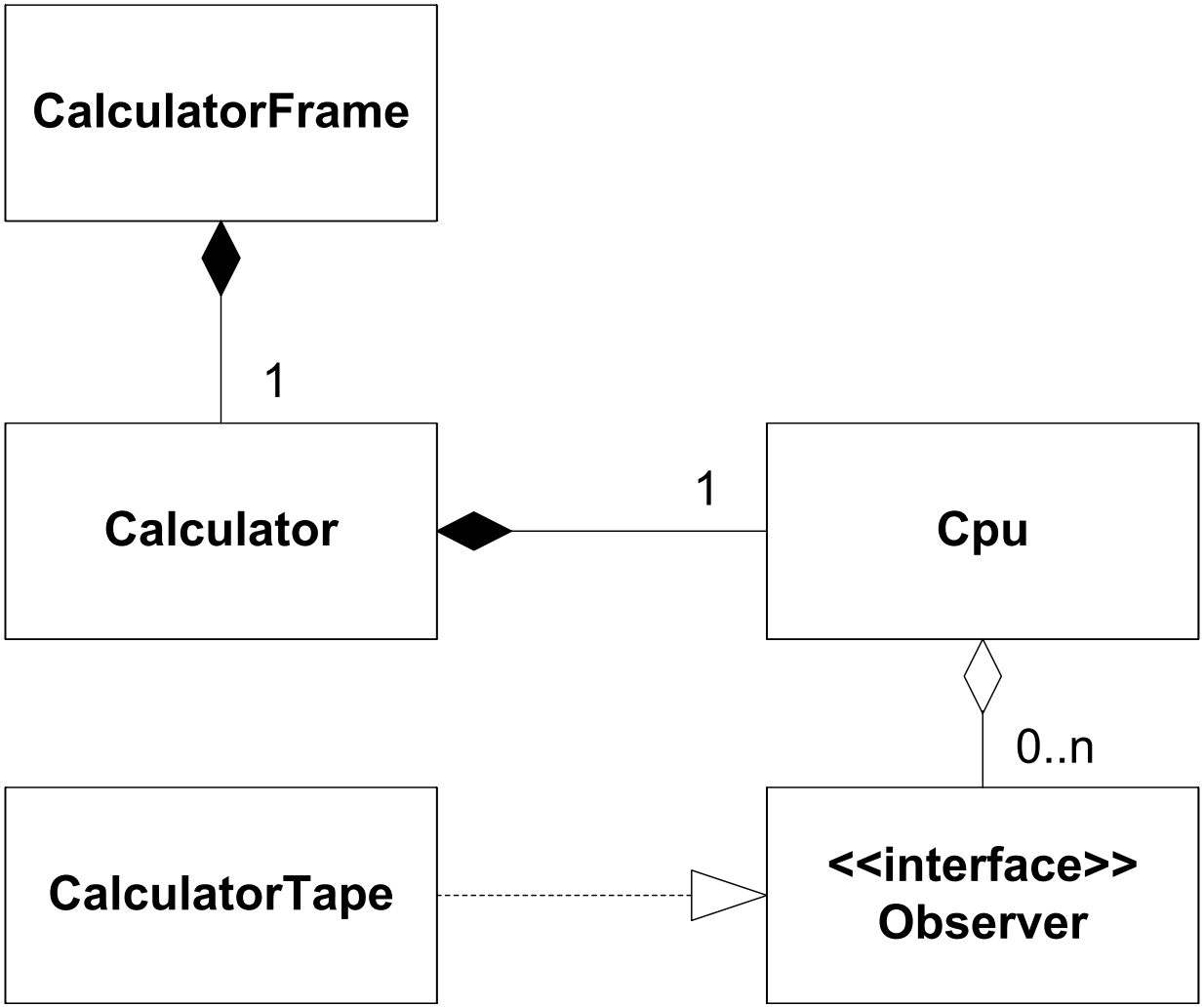


# Calculator Model Package

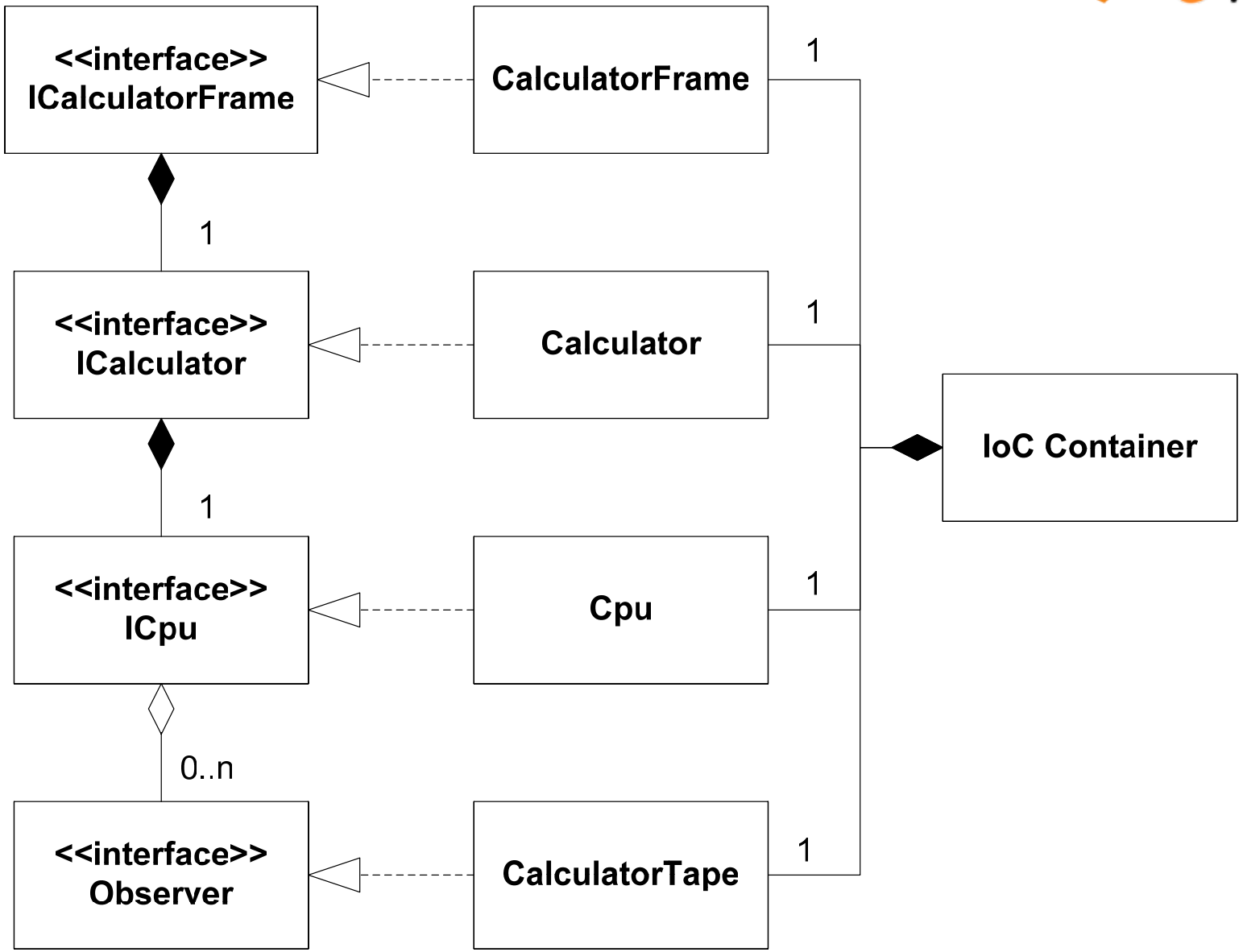



Created with Poseidon for UML Community Edition. Not for Commercial Use.

# Example Application



# Example After Dependency Injection



# Exercises

- **All exercises come as self contained Eclipse projects**
  - Breaking one exercise should not affect other exercises
  - Alternatively, all exercises can be run from the command line using Ant
- **Exercises cover the following scenarios**
  - Running the plain vanilla calculator
  - Using constructor based dependency injection
  - Using setter based dependency injection
  - Injecting stub implementations of components
- **Exercises encompass all containers discussed**
- **All examples coming with the IoC containers used are contained in the session material for all the eager beavers who finish the exercises well ahead of schedule**

# References

- **Containers (in alphabetical order)**
  - http://jakarta.apache.org/hivemind/
  - http://www.jboss.com/products/jbossmc
  - http://nanocontainer.codehaus.org/Home
  - http://www.picocontainer.org/Home
  - http://springframework.org/
    - http://www.springframework.org/docs/reference/index.html
- **Calculator**
  - http://www.objectsbydesign.com/projects/calc/overview.html
- **Other resources**
  - http://java-source.net/open-source/containers
  - http://www.martinfowler.com/articles/injection.html

# Contact Details

- **First of all, many thanks for attending our session!**
- **We sincerely hope you enjoyed the IoC workshop.**

- **If you have any questions or comments, please contact us at:**
  - wolf.schlegel@valtech.co.uk
  - eoin.woods@ubs.com
  - eoin@copse.org.uk