Dependency Injection using

By
Subbu Ramanathan

Presented at
The Great Lakes Software Symposium
Chicago, IL – October 3, 2004

Agenda

- The need for Pico Container
- Dependency Injection – What & How?
- Pico Container basics
- Container Hierarchies
- Life cycle support in Pico Container
- NanoContainer
- NanoWar
- Comparison with Spring
- Comparison with Avalon
- EJB3 ?
- Conclusion

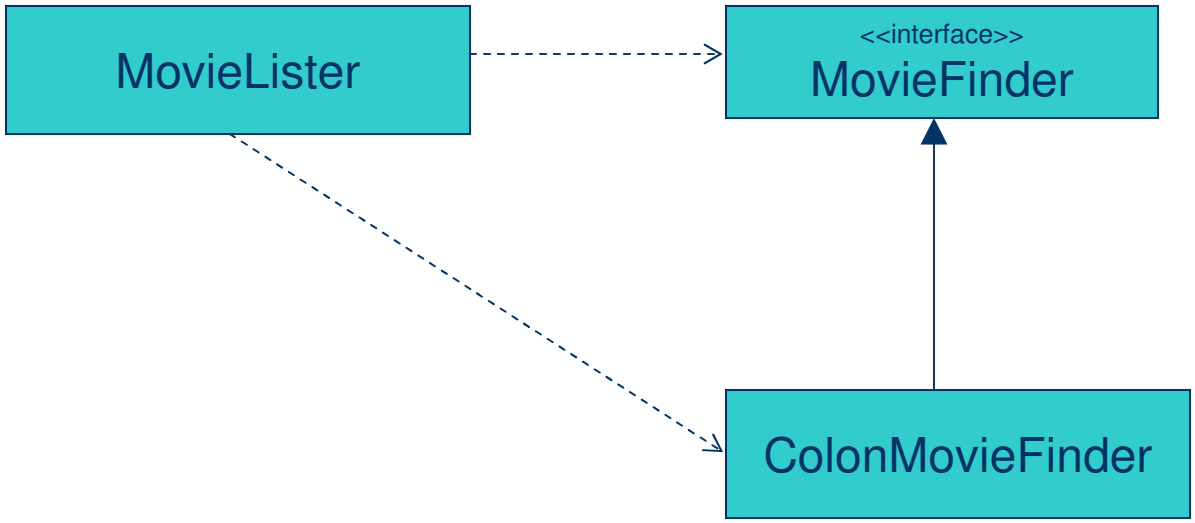
Coupling

- Best practice - create loosely coupled components
  - High Level Modules should not depend upon low level modules and vice-versa. Both should depend on abstractions.
  - Abstractions should not depend on details. Details must depend on abstractions.

Achieve loose coupling

- This goal must be accomplished
  - With minimal effort
  - Using a least intrusive mechanism
  - Without adverse functional implications

An Example



GOF Solution – Factory Pattern

- Use a Factory class to instantiate the component you need
- Only the Factory knows about the concrete class, not your application
- Pros
  - Helps to resolve tight coupling
- Cons
  - Intrusive: Components need to be requested explicitly
  - Need recompiling to change specific component in application

Alternative - Hollywood Principal

- aka “don’t call us, we’ll call you”
  - Components should not be explicitly requested
  - Components will be provided as needed
- aka Inversion of Control

Dependency Injection

- When IoC is about looking up plugin implementation, Martin Fowler calls it Dependency injection
- Dependency injection is a term used to resolve component dependencies by injecting an instantiated component to satisfy a dependency

Forms of Dependency Injection

- Constructor Injection with Pico Container
- Setter Injection with Spring
- Interface Injection with Avalon (Merlin)

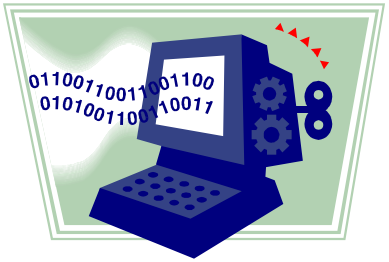
Constructor Injection

- Constructor Injection is a Dependency Injection variant where an object gets all its dependencies via the constructor.

Example Changed For Pico Container

- Pico Container uses a constructor to decide how to inject a finder implementation into the lister class.
- For this to work, the movie lister class needs to declare a constructor that includes everything it needs injected.

Example Using Pico Container



Constructor Injection Advantages

- It makes a strong dependency contract. Setting is atomic in a sense that either all or none of the dependencies are set and that it can occur once and only once
- Dependency is easily identifiable
- A dependency may be made immutable by making the dependency reference *final*

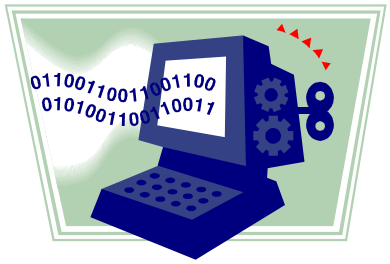
What is Pico Container?

- Light weight container
- Not a replacement for J2EE
- Non-intrusive. Components can be POJOs
- Provides component's life cycle management
- Extensible
- 50K jar that depends only on JDK1.3

Singleton vs. Pico

- Singleton provides one instance with global visibility
- Singleton causes hard-wired dependency
- Singletons are hard to test with Mock objects
- Container hierarchies in Pico provide fine grained control over a component's visibility scope
- A container (and its registered components) can get access to components registered in a parent container, but not vice-versa.

Container Hierarchies



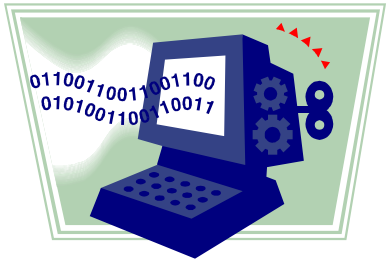
Lifecycle support in Pico

- Lifecycle involves post creation life of a component
- Useful when tree of component dependencies exist
- A “lifecycle method” invoked on parent container cascades through container components & down to child containers
- A “lifecycle method” invoked on a child container does NOT cascade up through parents
- Child containers must register as a component with parent container for lifecycle support to work

Lifecycle Interfaces

- Pico provides Startable & Disposable interfaces
- Startable two methods: start() and stop()
- Disposable has one method: dispose()
- It is okay if some or none of the components implement these lifecycle interfaces

Example Using Lifecycle



Lifecycle Appropriateness

- Developing unattended services
- A single point of control for multiple cascaded operations
- Transaction processing

Custom Lifecycle

- Instead of mandating the default lifecycle interfaces, Pico allows for development of custom Lifecycle support
- This functionality has been moved into the NanoContainer

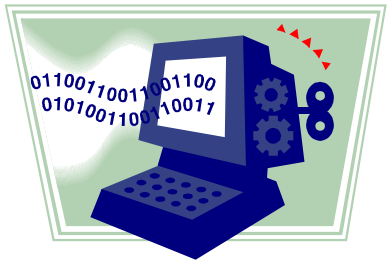
Custom Lifecycle Code

```
// Define the Lifecycle interface
public interface XTransaction {
    public void commit();
}

// Register Lifecycle implementer
PicoContainer pico = new DefaultPicoContainer();
Pico.registerComponentImplementation( XTransactionImpl.class);

// Invoke Lifecycle method
XTransaction xObj = (XTransaction) Multicaster.object(pico, true,
    new StandardProxyFactory());
xObj.commit();
```

NanoContainer

- Is a script enabled front-end for Pico Container
- Register components through a script
- Supports
  - Groovy
  - BeanShell
  - Jython
  - Rhino JavaScript
  - XML

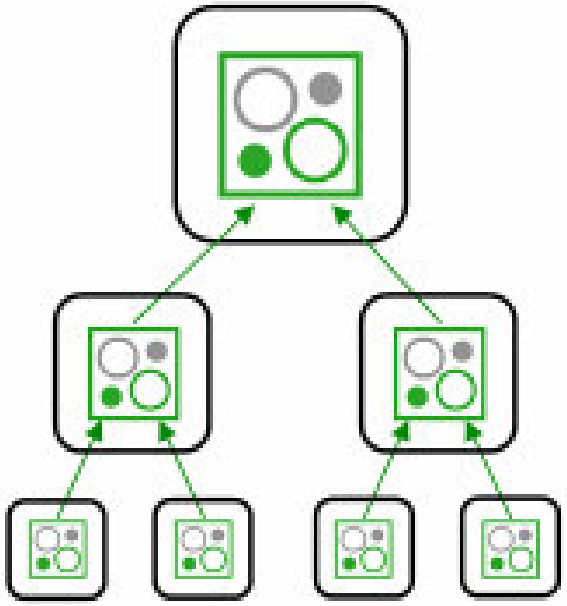
Example Using NanoContainer



NanoWar

- NanoWar allows to embed NanoContainer within a Java Servlet container
- It is non-intrusively activated in a Web app via a Servlet container listener
- Creates PicoContainer instances at the application, session, and request levels
- These are created when web application starts, session is created, and request is created respectively

NanoWar – Pico Containers



Web app scope
(javax.servlet.ServletContext)

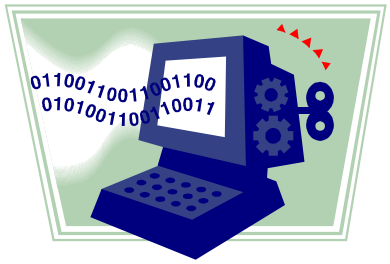
Session scope
(javax.servlet.http.HttpSession)

Request scope
(javax.servlet.ServletRequest)

NanoWar & WebWork

- The NanoWar – WebWork integration kit allows WebWork actions to be PicoContainer components
- Provides for decoupling Actions from Persistence Layer/Communication Layer
- Allows for easier unit testing of Actions outside of the web application using mock objects

Example Using NanoWar - WebWork



Nanocontainer & Persistence

- Nanocontainer-Hibernate
- Nanocontainer-Jaxor

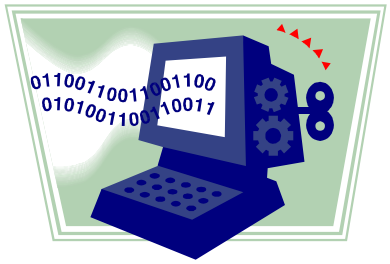
Pico Container & Jaxor

- Jaxor is an OR mapping tool which generates the persistence classes from XML based entity definitions
- Code generation is done using Velocity templates
- Jaxor can use Pico Container to register all concrete classes

Jaxor Classes

- Implementations of the interfaces that JaxorContextImpl depends on
- JaxorContainer provides lifecycle support by implementing Pico's Stoppable
- When the Container is Stopped, the Context is ended and db operations can be committed

Jaxor Code



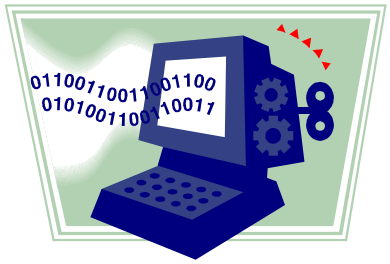
Other Pico Features

- Pico container also supports Setter Injection using SetterInjectionComponentAdapter class
- But, Pico really recommends Constructor Injection for obvious reasons

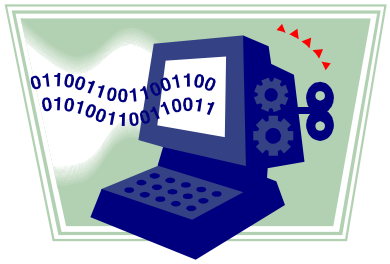
Setter Injection using Spring

- Spring framework uses a BeanFactory that stores information about all the beans used by the application
- The BeanFactory supplies the application with the needed beans when requested
- XMLBeanFactory is a BeanFactory that reads bean definitions from an XML file
- Beans can be defined as Singletons or not
- Spring can resolve bean dependencies

Spring Configuration



Spring Code



Spring – Constructor Injection

- Configuration XML

```
<bean id="hibernateClinic" class="samples.HibernatePetClinic">
    <constructor-arg index="0">
      <ref bean="sessionFactory"/>
  </constructor-arg>
</bean>
```

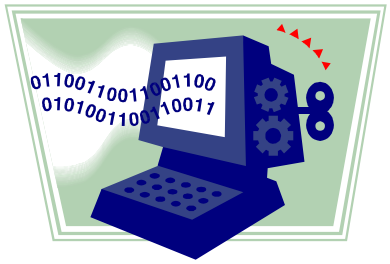
- Java Code

```
public class HibernatePetClinic implements PetClinic {
    public HibernatePetClinic(SessionFactory factory) {
                this.sessionFactory = factory;
    }
}
```

Interface Injection

- Injection is done through an interface
- The interface will define the injection method and the implementation class has to implement this interface and provide concrete implementation for the injection method.
- Avalon provides Interface Injection

Avalon Code



Dependency Injection with EJB3

```
@Session public class MyBean {
    private DataSource customerDB;

    @Inject private void setCustomerDB(DataSource customerDB) {
        this.customerDB = customerDB;
    }

    public void foo() {
        ...
        Connection c = customerDB.getConnection(); ...
    }
}
```

EJB3

- EJB 3 has some built-in support for dependency management using JNDI
- EJB3 relies on an application server

Solution Comparisons

- Really comes down to Pico Vs. Spring
- Spring is more mature than Pico
- Pico is much lighter than Spring. Spring is more than just an IoC container. It is a web framework as well.
- If you would like to use Spring for other reasons as well, Spring is the choice
- If you just need only IoC (or maybe for a non-web or a Struts/WebWork app), then Pico is the choice

Design Considerations

- What classes to register with container?
- Enforcing consistent standards
- Application extensibility & reusability?
- NanoContainer vs. PicoContainer
- Pico Stability
- Pico Support

Thank You

...

For
Attending
This
Presentation

(Please remember to fill out the session evaluation)