Dependency Injection and Custom Java Code Annotations in Eclipse 4 RCP

Shameless Plug Time

Please thank those that made this possible

- Location Sponsor: Cohesion*Force, Inc.*

Who am I?

Should you really listen to this Kyle Newman character?

- Two years experience using Eclipse RCP on Department of Defense program
- Primarily work on User Interface
- Battle daily with:
  - EMF, OSGi, RCP, E4 and Jface
- Background in web development/design, other tech and non-tech

Is this for me?

At what level of understanding is this aimed?

- Intermediate
  - Knowledge of Java development and a basic understanding of the Eclipse for Rich Client Platform will be useful in understanding concepts presented.

Is this for me (pt. 2)?

What is included in the presentation?

- What is Dependency Injection (DI)?
- What are Annotations?
- How does Eclipse 4 use DI?
- How do you define a custom annotation?

What is **NOT** DI?

Contrasting with Direct Dependency

- Hard dependencies on other classes
- **new**-ing your own objects

```
import org.efd.example.MyObject;

public class MyClass {

   public MyClass() {

      MyObject obj = new
MyObject();

   }

}
```

What is DI?
Quick and dirty definitions

- Inversion of control
- No direct object creation
- Ability to change dependencies at run-time or compile-time.
- “Dependency injection means giving an object its instance variables. Really. That's it.” - James Shore [http://bit.ly/1cUUGaA]

What is DI?

I/F for examples of coupled and manually injected dependencies

```java
public interface IOnlineBrokerageService {

    String[] getStockSymbols();

    double getBidPrice(String stockSymbol);

    double getAskPrice(String stockSymbol);

    void putBuyOrder(String stockSymbol, int shares, double buyPrice);

    void putSellOrder(String stockSymbol, int shares, double sellPrice);

}

public interface IStockAnalysisService {

    double getEstimatedValue(String stockSymbol);

}

public interface IAutomatedStockTrader {

    void executeTrades();

}
```

What is DI?
Example of highly coupled dependency

```java
public class VerySimpleStockTraderImpl implements IAutomatedStockTrader {

    private IStockAnalysisService analysisService = new StockAnalysisServiceImpl();

    private IOnlineBrokerageService brokerageService = new NYStockExchangeBrokerageServiceImpl();

    public void executeTrades() {

        ….

    }

}

public class MyApplication {

    public static void main(String[] args) {

        IAutomatedStockTrader stockTrader = new VerySimpleStockTraderImpl();

        stockTrader.executeTrades();

    }

}
```

What is DI?

Example of manually injected dependency

```
public class VerySimpleStockTraderImpl implements IAutomatedStockTrader {
    private IStockAnalysisService analysisService;
    private IOnlineBrokerageService brokerageService;
    public VerySimpleStockTraderImpl(
            IStockAnalysisService analysisService,
            IOnlineBrokerageService brokerageService) {
        this.analysisService = analysisService;
        this.brokerageService = brokerageService;
    }
    public void executeTrades() {
    }
}
public class MyApplication {
    public static void main(String[] args) {
        IStockAnalysisService analysisService = new StockAnalysisServiceImpl();
        IOnlineBrokerageService brokerageService = new
NYStockExchangeBrokerageServiceImpl();
        IAutomatedStockTrader stockTrader = new VerySimpleStockTraderImpl(analysisService,
brokerageService);
        stockTrader.executeTrades();
    }
}
```

What is DI?

Dependency Injection via Declarative Services

- E4 uses Declarative Services (DS) to provide OSGi services to non-application classes.
- DS is a component model that simplifies the creation of components that publish and/or reference OSGi Services.
- Core components of DS.
  - Producer component.xml
  - Producer manifest.mf
  - Consumer component.xml

What is DI?

Example of producer manifest.mf

```
Manifest-Version: 1.0

Bundle-ManifestVersion: 2

Bundle-Name: Myservice

Bundle-SymbolicName: org.hefdg.myservice

Bundle-Version: 1.0.4

Bundle-RequiredExecutionEnvironment: JavaSE-1.6

Import-Package: org.hefdg.myservice

Service-Component: OSGI-INF/component.xml
```

What is DI?

Example of producer component.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0"
name="IMyService">
   <implementation class="org.hefdg.myservice.MyServiceImpl"/>
   <service>
      <provide interface="org.hefdg.myservice.model.IMyService"/>
   </service>
</scr:component>
```

What is DI?

Example of consumer component.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0"
activate="activate" name="org.hefdg.consumer.component">
   <implementation class="org.hefdg.consumer.impl.ConsumerImpl"/>
   <reference bind="bindMyService" cardinality="1..1"
   interface="org.hefdg.myservice.model.IMyService"
   name="MyServiceInterface" policy="static" unbind="unbindMyService" />
</scr:component>
```

What is DI?

Example of code using declarative service

```
package org.hefdg.consumer.impl.ConsumerImpl;

import org.hefdg.myservice.model.IMyService;

public class ConsumerImpl {

  private IMyService service;

  public void stuff() {

    System.out.println(service.getStuff());

  }

  public synchronized void bindMyService(IMyService service) {  // Method used by DS to set the service

    this.service = service; // Service was set. Thank you DS!

  }

  public synchronized void unbindMyService(IMyService service) {  // Method used by DS to unset the service

    if (this.service == service)

      this.service = null;

  }

}
```

What is DI?
Dependency Injection via Annotations

- E4 also uses Annotations to inject objects and services into the application

What are Annotations?

- Syntactic metadata
- Added to Java source code
- Retained by JVM for run-time retrieval
- Since Java 1.5

Java Annotations

Built-in annotations that can be applied

To code:

- @Override
- @Deprecated
- @SuppressWarnings

To annotations:

- @Retention
- @Documented
- @Target
- @Inherited

Java Annotations

Create your own for POJO use

- Similar to normal interface declarations
- Method declaration defines an element of the annotation type.
- No parameters or throws clause
- Return types are restricted to primitives
- Methods can have default values

```
// @Twizzle is an annotation to method
toggle()

@Twizzle

public void toggle() {...}

// Declares the annotation Twizzle.

public @interface Twizzle {}
```

E4 Annotations

Stand on the shoulders of giants

Standard

- @Inject
- @Named
- @Singleton
- @PostConstruct
- @PreDestroy

E4AP-specific

- @Optional
- @Active
- @Preference
- @Creatable
- @CanExecute, @Execute
- @Focus
- @AboutToShow, @AboutToHide
- @GroupUpdates
- @EventTopic, @UIEventTopic

How does E4 use DI?

Examples of using annotations for injection

- Hierarchy of call order
  - **@Inject** Constructor Call
  - **@Inject** Field initialization
  - **@Inject** Method call
  - **@PostConstruct** Method call

How does E4 use DI?

Injecting objects through the constructor

- With this, the Composite used to create the part is injected as a constructor parameter.

```
public class MyView
{
   @Inject
   public MyView(Composite parent)
   {
      // Implement the View placed
      // on the Parent
   }


}
```

How does E4 use DI?

Inject services at the field level

- The Logger must be injected or the parent will throw an Exception when created.
- By using @Optional, the RandomService is not required.

```
@Inject
private Logger logger;


@Inject
@Optional
private RandomService randomService;
```

How does E4 use DI?

Objects selected via ESelection Service

- This example shows how E4 can re-inject values if they are changed.
- With this, applications are freed from installing/removing listeners.

```
@Inject
public void printSelection(@Optional
      @Named(IServiceConstants.ACTIVE_SELECTION)
      Object object) {
   if (object != null) {
      // Print out the active selection object
      System.out.println(object);
   }
}
```

How does E4 use DI?
Behavior Annotations for GUI interaction

- Standard Annotations for creating MParts

```
public class MyPart {

    @PostConstruct
    public void postConstruct(Composite parent) {
    }

    @Focus
    public void setFocus(Composite parent) {
    }

    @PreDestroy
    public void preDestroy(MWindow mWindow) {
    }

    @Persists
    public void persists(MDirtyable mDirtyable) {
    }

    @PersistState
    public void persistState(MyPersistenceService service) {
    }
}
```

How does E4 use DI?

Use @Named with predefined IServiceConstants

- ACTIVE_SELECTION
- ACTIVE_CONTEXTS
- ACTIVE_PART
- ACTIVE_SHELL

```
@Inject

public void printMyContexts(@Optional


@Named(IServiceConstants.ACTIVE_CONTEXTS)
Object object) {

   if (object != null) {

      // Print out the active selection
object

      System.out.println(object);

  }

}
```

How does E4 use DI?

Use @Named to get IEclipseContext variables

- Useful for watching for change in singleton-like variables

```
@PostConstruct

public void postConstuct(IEclipseContext
context) {

   context.set("value", null);

   // ...

   context.set("value", "MyValue");

}

@Inject

public void valueChanged(@Optional
@Named("value") Object object) {

   // ...

}
```

How does E4 use DI?

Use @CanExecute and @Execute to make functionality available

- Available in application model via Handler/ Command
- Available programmatically via EHandlerService and ECommandService

```
public class ExitHandler {

  @Execute

  public void execute(IWorkbench workbench) {

    workbench.close();

  }

  // NOT REQUIRED IN THIS EXAMPLE

  // just to demonstrates the usage of

  // the annotation

  @CanExecute

  public boolean canExecute() {

    return true;

  }

}
```

How does E4 use DI?

Use @EventTopic and @UIEventTopic in events

- Alert active Parts of events
- Pass objects to active Parts

```
@Inject

private IEventBroker eventBroker;

// asynchronously

broker.post(MyEventConstants.TOPIC_NEW, "New data");

// synchronously - calling code is blocked until delivery

broker.send(MyEventConstants.TOPIC_NEW, myObject);

@Inject

@Optional

private void
getNotified(@UIEventTopic(MyEventConstants.TOPIC_UPDATE)

    String s) {

  // Do something with myObject

}
```

How do you define a custom annotation?

Part 1: Define the annotation in an interface

```java
package org.hefdg.customannotation;

import java.lang.annotation.Documented;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

@javax.inject.Qualifier

@Documented

@Target({ElementType.PARAMETER, ElementType.FIELD})

@Retention(RetentionPolicy.RUNTIME)

public @interface MyAnnotation {

}
```

How do you define a custom annotation?

Part 2: Define the annotation processor in OSGi service

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0"
  name="org.hefdg.customannotation">
   <implementation
     class="org.hefdg.customannotation.UniqueMyObjectSupplier"/>
   <service>
      <provide interface="org.eclipse.e4.core.di.suppliers.ExtendedObjectSupplier"/>
   </service>
   <property name="dependency.injection.annotation" type="String"
     value="org.hefdg.customannotation.UniqueTodo"/>
</scr:component>
```

How do you define a custom annotation?

Part 3: Define the annotation processor in OSGi service

```java
package com.example.e4.rcp.todo.ownannotation.internal;

import org.eclipse.e4.core.di.suppliers.ExtendedObjectSupplier;

import org.eclipse.e4.core.di.suppliers.IObjectDescriptor;

import org.eclipse.e4.core.di.suppliers.IRequestor;

import org.hefdg.myservice.model.IMyObject;

public class UniqueMyObjectSupplier extends ExtendedObjectSupplier {

@Override

public Object get(IObjectDescriptor descriptor, IRequestor requestor,

         boolean track, boolean group) {

      // for the purpose of providing a simple example here, we return a hard-coded

      MyObject myObject = new MyObject();

      return myObject;

   }

}
```

How do you define a custom annotation?

Part 4: Using your custom annotation in code

- Add annotation to a field or method parameter in a part

```
@Inject

public void
setMyObject(@MyAnnotation MyObject
myObject) {

   // do something with the _unique_

   // MyObject

}
```

How do you define a custom annotation?

Caveats on creating custom annotations for use with E4

- Extended object suppliers have no access to IEclipseContext
- Limited to search for objects independent of Eclipse context
- i.e. Preferences are extended object suppliers
  - Look for the preference values on the file system

What's coming up next?

Please plan to attend our upcoming events

- April 16 - “How to use Hudson”
  - Presented by J. Langley

Standard Annotations

E4AP's injector is based on the standard JSR 330 annotations

@Inject	Marks a constructor, method, or field as being available for injection.
@Named	Multiple objects can be distinguished by providing a name, both on setting them as well as requesting them for injection.
@Singleton	Indicates the class should only be instantiated once per injection scope. Typical E4AP applications have only a single injector scope for the application.
@PostConstruct	Provides lifecycle notification for created objects. All methods annotated with @PostConstruct are called after an object has been fully injected.
@PreDestroy	Provides lifecycle notification for created objects. All methods annotated with @PreDestroy are called before an object is to be uninjected and released.

E4 Specific Annotations

Page 1

@Optional	Can be applied to methods, fields, and parameters to mark them as optional for the dependency injection. If this annotation is specified, then if injector is unable to find a value: • for parameters: a null value will be injected; • for methods: the method calls will be skipped; • for fields: the values will not be injected.
@Active	Indicates the the value should be resolved from the active context.
@Preference	Provides simple interfacing with the Eclipse preferences framework.
@Creatable	Automatically created by the injector if an instance was not present in the injection context

E4 Specific Annotations
Page 2

@CanExecute	Tags methods that should be executed for a command handler. Should return a boolean.
@Execute	Tags methods that should be executed for a command handler.
@Focus	Used on a method to be called when the part receives focus. Parts must implement this method in such a way that a child control of their part can receive focus.
@AboutToShow	Used in dynamic menu contribution elements. The respective annotated methods are called on showing of the menu, and on hiding of the menu. An empty list is injected. Do not put long-running code here. It delays the opening process of the menu.
@AboutToHide	Used in dynamic menu contribution elements. The respective annotated methods are called on showing of the menu, and on hiding of the menu. Injected with the list from *@AboutToShow*, containing the elements contributed in *@AboutToShow*

E4 Specific Annotations
Page 3

@GroupUpdates	Indicates to the framework that updates should be batched.
@EventTopic	Tags methods and fields that should be notified on event changes. Both the event's DATA object and the actual OSGi Event object (org.osgi.service.event.Event) are available.
@UIEventTopic	Tags methods and fields that should be notified on event changes. Ensures the event notification is performed in the UI thread. Both the event's DATA object and the actual OSGi Event object (org.osgi.service.event.Event) are available.
@Persists	Called if a save request on the Part is triggered. Used by the part service to identify the method to call if a save is triggered via this service.
@PersistState	Called before the model object is disposed, so that the part is able to save its instance state. Also called before the method annotated with @PreDestroy is called.

Further Reading

- http://grepcode.com/file/repository.grepcode.com/java/eclipse.org/4.2/org.eclipse.e4.ui/services/0.10.1/org/eclipse/e4/ui/services/IServiceConstants.java
- http://www.eclipsecon.org/2013/sites/eclipsecon.org.2013/files/E4_Injection_OPCoach_talk_0.pdf
- http://en.wikipedia.org/wiki/Java_annotation
- https://wiki.eclipse.org/Eclipse4/RCP/Dependency_Injection
- http://eclipsesource.com/blogs/tutorials/eclipse-4-e4-tutorial-part-6-behavior-annotations/
- http://www.vogella.com/tutorials/EclipseRCP/article.html
- http://www.vogella.com/tutorials/Eclipse4EventSystem/article.html

Code/Markup Examples

- http://en.wikipedia.org/wiki/Java_annotation
- http://www.vogella.com/tutorials/OSGiServices/article.html
- http://en.wikipedia.org/wiki/Dependency_injection

DI Frameworks

- Spring
- Google Guice
- Glassfish H2K
- Microsoft Managed Extensibility Framework
- PicoContainer