# Start container in Java SE

## ferris-cdi-initialized

* Weld-proprietary solution
* pom.xml
  + Weld StartMain class
  + Assembly plugin for single jar
* beans.xml
* Problems
  + Weld-specific class in pom.xml
  + Weld-specific classes in ContainerInitializedObserver

public class ContainerInitializedObserver {

public void bootstrapped(@Observes ContainerInitialized event, @Parameters List<String> parameters) {

System.out.printf(">>> ENTER ContainerInitializedObserver#bootstrapped\n");

}

}

## ferris-cdi-bootstrap

* Standard solution
  + CDI 2.0 proposal
* No Weld-specific classes

public class Main {

public static void main(String[] args) {

System.out.printf(">>> ENTER Main#main\n");

try (CDI<Object> cdi = CDI.getCDIProvider().initialize()) {

// start the container, retrieve a bean and do work with it

System.out.printf(">>> CDI bootstrap complete. \n");

} catch (Exception e) {

e.printStackTrace();

}

}

}

# Basic (DI)

## ferris-cdi-research

* pom.xml
* log4j.properties
* org.ferris.cdi.research.basics

## Field injection

* PropertyInjectionExample

## Constructor injection

* ConstructionInjectionExample
* Only one constructor with @Inject !!

## Method injection (Initializer/Setter method)

* MethodInjectionExample
* There can be multiple within the bean

## A class can use all 3 at the same time

* AllInjectionExample

# Interfaces and classes

## ferris-cdi-research

* org.ferris.cdi.research.interfaces

## Use the interface with @Inject

public interface HelloWorldService {

String getHello();

}

public class HelloWorldServiceBean implements HelloWorldService {

@Override

public String getHello() {

return "Hello Bean!";

}

}

public class HelloWorldExample {

@Inject

HelloWorldService hwService;

public void sayHello() {

System.out.printf(">>>> %s\n", hwService.getHello());

}

}

# Shared project

## ferris-cdi-shared

* beans.xml
* SharedService

## ferris-cdi-research

* org.ferris.cdi.research.sharedproject

public class SharedProjectExample {

@Inject

private SharedService sharedService;

public void printName() {

System.out.printf(

">>>> SharedService name = \"%s\"\n", sharedService.getName());

}

}

# Lifecycle callbacks

## ferris-cdi-research

* org.ferris.cdi.research.lifecycle

## @PreDestroy @PostConstruct

public class LifecyleExample {

@PostConstruct

protected void postConstruct() {

System.out.printf(">>>> LifeCycleExample Post Construct\n");

}

@PreDestroy

protected void preDestory() {

System.out.printf(">>>> LifeCycleExample Pre Destroy\n");

}

}

# Deployment Exceptions

## ferris-cdi-research

* org.ferris.cdi.research.exception

## Unsatisfied exception

* Interface but no implementation

public interface UnsatisfiedImplementation {}

public class UnsatisfiedExample {

@Inject

private UnsatisfiedImplementation missingImplementation;

}

## Ambiguous exception

* Multiple implementations

public interface AmbiguousService {}

public class AmbiguousServiceA implements AmbiguousService {}

public class AmbiguousServiceB implements AmbiguousService {}

public class AmbiguousExample {

@Inject

private AmbiguousService service;

}

## Vetoed

* @Vetoed
* All \*Example classes
* Need this so CDI bootstrap won’t fail

# Qualifiers

## ferris-cdi-research

* org.ferris.cdi.research.qualifiers

## Type-safe way to resolve ambiguity

* Annotate the implementations
* Use qualifier along with @Inject

## Service classes

public interface OrderService {

void save(Object o);

}

**//@DB**

**@Storage(value=DB)**

public class DatabaseOrderService implements OrderService {

@Override

public void save(Object o) {

out.printf(">>>> @Storage save to database\n");

}

}

**//@JMS**

**@Storage(value=JMS)**

public class JMSOrderService implements OrderService {

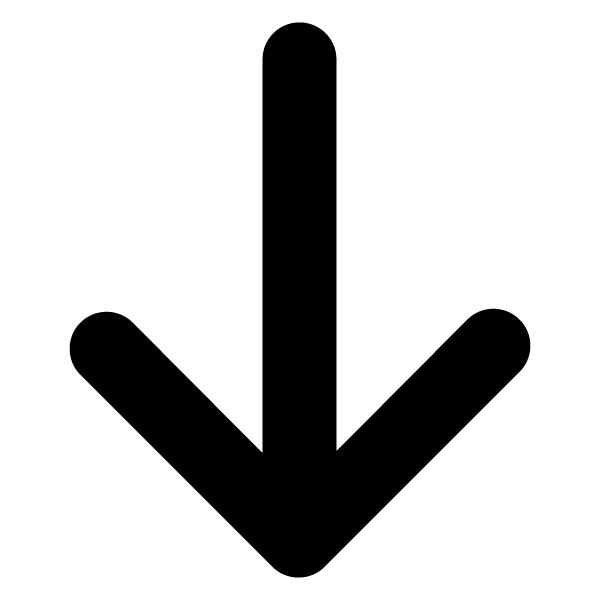
@Override

public void save(Object o) {

out.printf(">>>> @Storage post Order to JMS Queue\n");

}

}



## Example classes

public class DatabaseExample {

@Inject

**//@DB**

**@Storage(value=DB, description="Save to DB")**

OrderService orderService;

public void save() {

System.out.printf(">>>> DatabaseExample %s\n", orderService.getClass().getName());

orderService.save(new Object());

}

}

public class JMSExample {

**...**

}

## Specific qualifiers

@Qualifier

@Retention(RetentionPolicy.RUNTIME)

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

public @interface DB {

}

## Generic qualifiers

@Qualifier

@Retention(RetentionPolicy.RUNTIME)

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

public @interface Storage {

public enum StorageType {

DB, JMS

}

StorageType value();

@Nonbinding

String description() default "";

}

# Producer/Disposer

## ferris-cdi-research

* org.ferris.cdi.research.producer

## Field producer

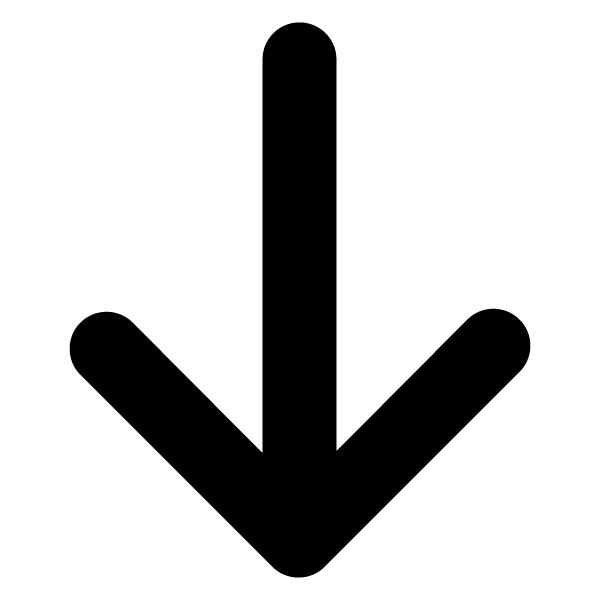
* IntegerListProducer, FieldProducerExample, FieldProducerMain

public class IntegerListProducer {

@Produces

private List<Integer> intList = new ArrayList(){{add(1); add(2); add(3);}};

}



## Method producer (no-argument)

* Get words from data file
  + “voice-test-one.txt”
* Word, WordListProducer, MethodProducerExample, MethodProducerMain

public class Word {

private String value;

public Word(String value) {

this.value = value;

}

@Override

public String toString() {

return value;

}

}

public class WordListProducer {

@Produces

public List<Word> produceWordList() {

List<Word> wordList = new ArrayList<>();

try (LineNumberReader reader

= new LineNumberReader(new FileReader("voice-test-one.txt"));

){

for (String line=reader.readLine(); line!=null; line=reader.readLine()) {

wordList.add(new Word(line));

}

} catch (IOException e) {

wordList.clear();

}

return wordList;

}

public void disposeWordList(@Disposes List<Word> wordList) {

System.out.printf(">>>> Disposing list!!!\n");

wordList.clear();

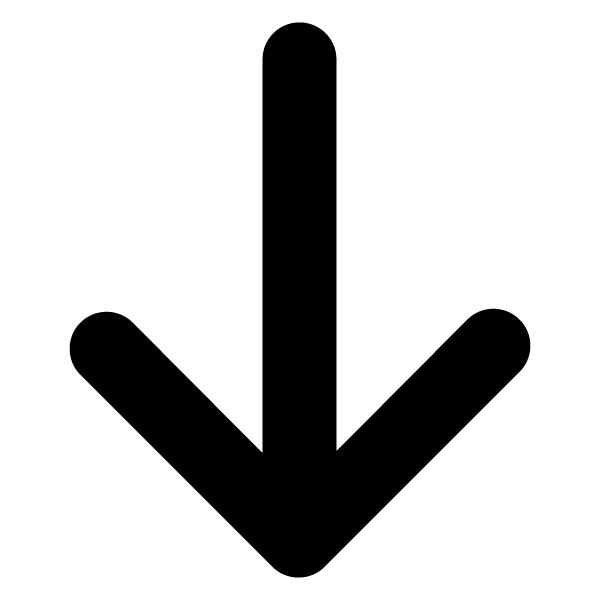
wordList = null;

}

}

## Method disposer

* See WordListProducer above



## Method producer (arguments)

* Shared project - shoppingcart

public class ShoppingCartServiceProducer {

@Produces

public ShoppingCartService produceShoppingCartService(

ShoppingCartErrorHandler err, ShoppingCartProcessHandler proc

) {

return new ShoppingCartService(err,proc);

}

}

## Producers with Qualifiers

* Go back to the qualifiers examples and add producers.

# InjectionPoint

## ferris-cdi-research

* org.ferris.cdi.research.injectionpoint

## Get declaring class

* Examples have used System.out.print
* Let’s use Logger

public class LoggerProducer {

@Produces

public Logger getLogger(InjectionPoint ip) {

String name = ip.getMember().getDeclaringClass().getName();

System.out.printf("|||| Get logger for %s\n",name);

return Logger.getLogger(name);

}

}

## Get annotations and qualifiers

* Get string from properties file by key
  + “messages.properties”

@Qualifier

@Retention(RetentionPolicy.RUNTIME)

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

public @interface Key {

@Nonbinding

String value() default "";

}

public class StringProducer {

private Properties props;

public StringProducer() throws Exception {

props = new Properties();

props.load(new FileInputStream("messages.properties"));

}

@Produces @Key

public String produceString(InjectionPoint ip) {

Key m = ip.getAnnotated().getAnnotation(Key.class);

return props.getProperty(m.value(), "-UNKNOWN-");

}

}

# Events

Event fired with the fireAsync() method is fired asynchronously. All resolved synchronous observers are called in the same thread in which fireAsync() was called, and all the resolved asynchronous observers are called in one or more different threads. If synchronous observer have to be notified, fireAsync returns immediately after the last synchronous observer has returned. Otherwise it returns immediately.

## ferris-cdi-research

* org.ferris.cdi.research.event

## Event

public class EventExample {

@Inject

private Logger log;

@Inject

private Event<AnEvent> event;

public void fire() {

log.debug("Create AnEvent");

AnEvent evnt = new AnEvent();

// log.debug("Before fire()");

// event.fire(evnt);

// log.debug("After fire()");

log.debug("Before fireAsync()");

event.fireAsync(evnt);

log.debug("After fireAsync()");

}

}

## Synchronous

* SyncObserver[A-D]

public class SyncObserverA {

@Inject

protected Logger log;

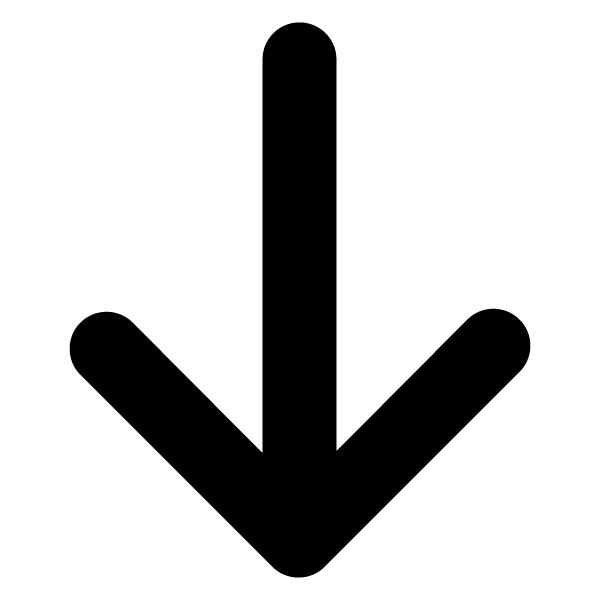
public void observes(@Observes **@Priority(AnEvent.Step1)** AnEvent evnt) {

log.debug(String.format(

"Thread: \"%s\"", Thread.currentThread().getName()));

}

}



## Priority

* And @Priority to the sync observers
* org.jboss.weld.experimental.Priority
  + It’s experimental!

## Asynchronous

* Basically just synchronous only off the main thread
  + Kinda sucks
  + Still a work in progess for CDI 2.0
* AsyncObserver[A-D]

public class AsyncObserverA {

@Inject

protected Logger log;

public void observes(@ObservesAsync **@Priority(AnEvent.Step5)** AnEvent evnt) {

try {

Thread.currentThread().sleep(1000 \* 3);

} catch (Exception e) {

e.printStackTrace();

}

log.debug(String.format(

"Thread: \"%s\"", Thread.currentThread().getName()));

}

}

# Interceptor

## ferris-cdi-research

* org.ferris.cdi.research.interceptor

## InterceptorBinding

@Inherited

@InterceptorBinding

@Retention(RetentionPolicy.RUNTIME)

@Target({ElementType.METHOD, ElementType.TYPE})

public @interface ExceptionRetry {

}

## Interceptor

**@ExceptionRetry**

@Interceptor

**@Priority(Interceptor.Priority.APPLICATION)**

public class ExceptionRetryInterceptor {

@Inject

protected Logger log;

@AroundInvoke

public Object retryIfExceptionCaught(InvocationContext ctx) throws Exception {

Exception caught = null;

for (int i = 1, imax = 4; i <= imax; i++) {

try {

return ctx.proceed();

} catch (Exception e) {

caught = e;

log.warn(String.format("Exception caught on attempt %d of %d", i, imax));

}

try {

Thread.sleep(1000 \* 3);

} catch (InterruptedException e) {

}

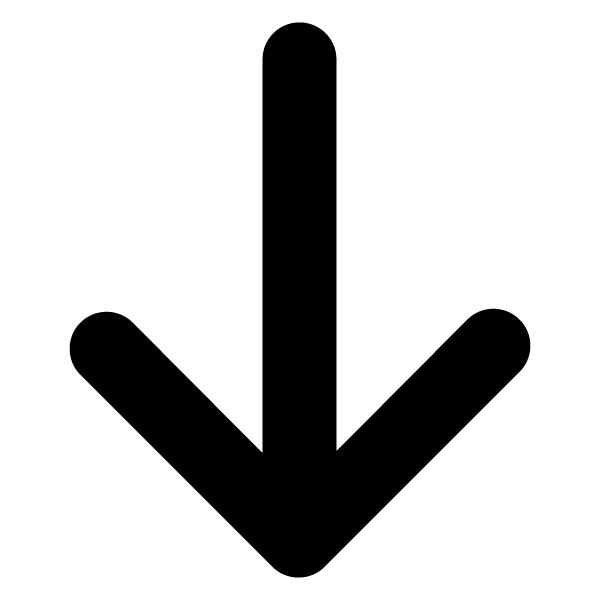
}

log.fatal("All retry attempts failed.", caught);

throw caught;

}

}



## Example

public class ExceptionRetryExample {

@Inject

protected Logger log;

private static int cnt = 1;

**@ExceptionRetry**

public void superImportant() {

log.debug("Trying superImportant()...");

if (cnt <= 3) {

throw new RuntimeException(

String.format("Super important failed on try #%d",cnt++));

}

cnt = 1;

log.debug("SUCCESS!");

}

@ExceptionRetry

public void lessImportant() {

log.debug("Trying lessImportant()...");

throw new RuntimeException(

String.format("Oops!"));

}

}

# Decorator

## ferris-cdi-research

* org.ferris.cdi.research.decorator

## Interface

* Must start with an interface

public interface TalkingService {

String sayOne();

String sayTwo();

String sayThree();

}

## Implementation Bean

* TalkingServiceBean

## Implementation Decorator

* Implement the same interface
* Make abstract to avoid implementing all the methods
* Use @Decorator to make it a decorator
* Use @Delegate to inject bean

**@Decorator**

**@Priority(Interceptor.Priority.APPLICATION)**

public abstract class TalkingServiceDecorator implements TalkingService {

@Inject @Delegate

TalkingService talkingService;

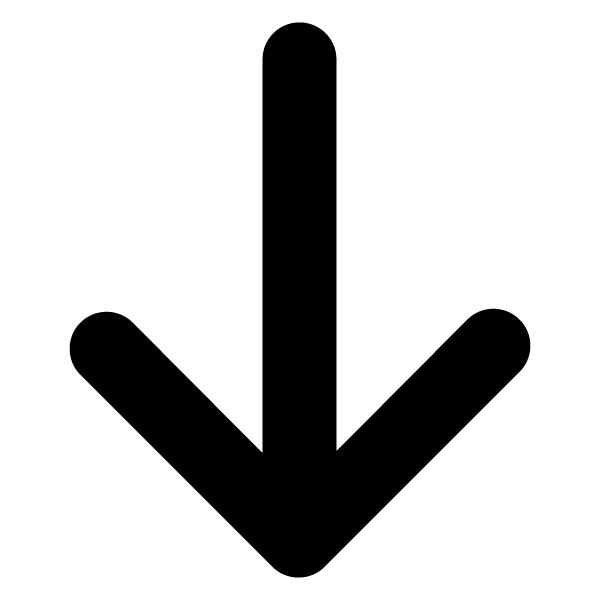
@Override

public String sayThree() {

return String.format("DECORATOR!!! %s", talkingService.sayThree());

}

}



## Example

* TalkingServiceExample
* User interface TalkingService to @Inject
* CDI will inject the decorator

public class TalkingServiceExample {

@Inject

protected Logger log;

@Inject

protected TalkingService talkingService;

public void talk() {

log.debug(talkingService.sayOne());

log.debug(talkingService.sayTwo());

log.debug(talkingService.sayThree());

}

}