CDI University

@antoine_sd
Red Hat, http://cdi-spec.org
CDI specification leader

DEVOXX™ France

Antoine Sabot-Durand

- Senior Software Developer @Red Hat

- Java & OSS :
  - CDI co-spec lead
  - CDI eco-system development
  - Tech Lead on Agorava

- @antoine_sd
- @cdispec

CDI basics

DEVOXX™

Context & Dependency Injection

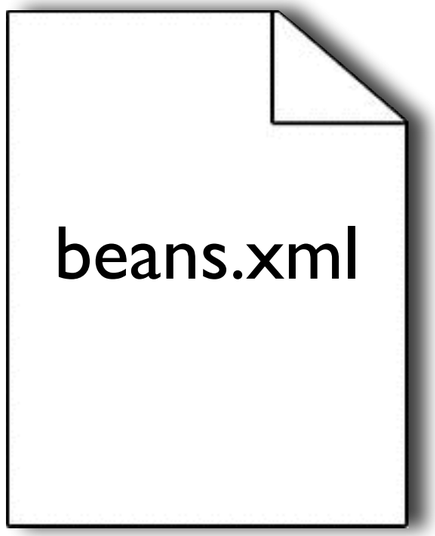
- CDI is a lot of things
  - Java EE IoC standard also working on Java SE
  - Context management
  - Observer pattern included
  - The « cement » between most Java EE spec
  - An evolution engine for Java EE thanks to portable extensions
  - All of this based on java strong typing

CDI history & actual state

- Main milestone :
  - December 2009 : CDI 1.0
  - June 2013 : CDI 1.1
  - April 2014 : CDI 1.2
  - Summer 2014 : CDI 2.0 development starting
- Implementations :
  - JBoss Weld (RI) : WildFly, JBoss EAP, Glassfish, Weblogic
  - Apache OpenWebBeans : TomEE, Websphere
- Check http://cdi-spec.org

CDI activated by default in Java EE 7

- In Java EE 6, you must add a beans.xml file to your archive
- No need to in Java EE 7 but you can still to have an explicit activation (behave differently)



Bean

- In Java EE 6 everything is a Managed Bean
  - Managed beans are basic components
  - They are managed by the container
  - They all have a lifecycle
  - They can be intercepted (AOP)
  - They can be injected
  - Accessible from outside CDI code.

Dependency Injection

```
@Inject
```

Dependency Injection

```
public class HelloService {
    public String hello() {
        return "Hello World!";
    }
}
```

Dependency Injection in constructor

```
public class MyBean {

    private HelloService service;

    @Inject
    public MyBean(HelloService service) {
        this.service = service;
    }
    public void displayHello() {
        display( service.hello();
    }
}
```

Dependency Injection in setter

```
public class MyBean {

    private HelloService service;

    @Inject
    public void setService(HelloService service) {
        this.service = service;
    }
    public void displayHello() {
        display( service.hello();
    }
}
```

Dependency Injection in field

```
public class MyBean {

    @Inject HelloService service;

    public void displayHello() {
        display( service.hello();
    }
}
```

Qualifiers

```
public interface HelloService {
    public String hello();
}
public class FrenchHelloService implements HelloService {
    public String hello() {
        return "Bonjour tout le monde!";
    }
}
public class EnglishHelloService implements HelloService {
    public String hello() {
        return "Hello World!";
    }
}
```

Qualifiers

```
@Qualifier
@Retention(RUNTIME)
@Target({FIELD, TYPE, METHOD, PARAMETER})
public @interface French {}

@Qualifier
@Retention(RUNTIME)
@Target({FIELD, TYPE, METHOD, PARAMETER})
public @interface English {}
```

Qualifiers

```
@French
public class FrenchHelloService implements HelloService {
    public String hello() {
        return "Bonjour tout le monde!";
    }
}

@English
public class EnglishHelloService implements HelloService {
    public String hello() {
        return "Hello World!";
    }
}
```

Qualifiers

```
public class MyBean {
    @Inject @French HelloService service;
    public void displayHello() {
        display( service.hello();
    }
}
public class MyBean {
    @Inject @English HelloService service;
    public void displayHello() {
        display( service.hello();
    }
}
```

Qualifiers with members

```
@Qualifier
@Retention(RUNTIME)
@Target({FIELD, TYPE, METHOD, PARAMETER})
public @interface Language {

    Languages value();

    @Nonbinding String description() default "";

    public enum Languages {
        FRENCH, ENGLISH
    }
}
```

Qualifiers

```
@Language(FRENCH)
public class FrenchHelloService implements HelloService {
    public String hello() {
        return "Bonjour tout le monde!";
    }
}
@Language(ENGLISH)
public class EnglishHelloService implements HelloService {
    public String hello() {
        return "Hello World!";
    }
}
```

Qualifiers

```
public class MyBean {
    @Inject @Language(ENGLISH) HelloService service;
    public void displayHello() {
        display( service.hello();
    }
}

public class MyBean {
    @Inject @Language(FRENCH) HelloService service;
    public void displayHello() {
        display( service.hello();
    }
}
```

Qualifiers

```
public class MyBean {
    @Inject @French
    HelloService service;
}

@French @Console @Secured
public class FrenchHelloService implements HelloService {

}
```

Qualifiers

```
public class MyBean {
    @Inject @French @Console
    HelloService service;
}

@French @Console @Secured
public class FrenchHelloService implements HelloService {

}
```

Qualifiers

```
public class MyBean {
    @Inject @French @Console @Secured
    HelloService service;
}

@French @Console @Secured
public class FrenchHelloService implements HelloService {

}
```

Qualifiers

```
public class MyBean {
     @Inject @French @Console @Secured
     HelloService service;
}

@French @Secured
public class FrenchHelloService implements HelloService {

}
```

Reserved Qualifiers

```
@Default
@Any
@Named
```

Programmatic lookup

```
public class MyBean {

    @Inject Instance<HelloService> service;

    public void displayHello() {
        display( service.get().hello() );
    }
}
```

Programmatic lookup

```
public class MyBean {

    @Inject Instance<HelloService> service;

    public void displayHello() {
        if (!service.isUnsatisfied()) {
            display( service.get().hello() );
        }
    }
}
```

Programmatic lookup

```
public class MyBean {

    @Inject @Any Instance<HelloService> services;

    public void displayHello() {
        for (HelloService service : services) {
            display( service.hello() );
        }
    }
}
```

Programmatic lookup

```
public class MyBean {

    @Inject @Any Instance<HelloService> services;

    public void displayHello() {
        display(
            service.select(
              new AnnotationLiteral()<French> {})
                .get() );
    }
}
```

Contexts

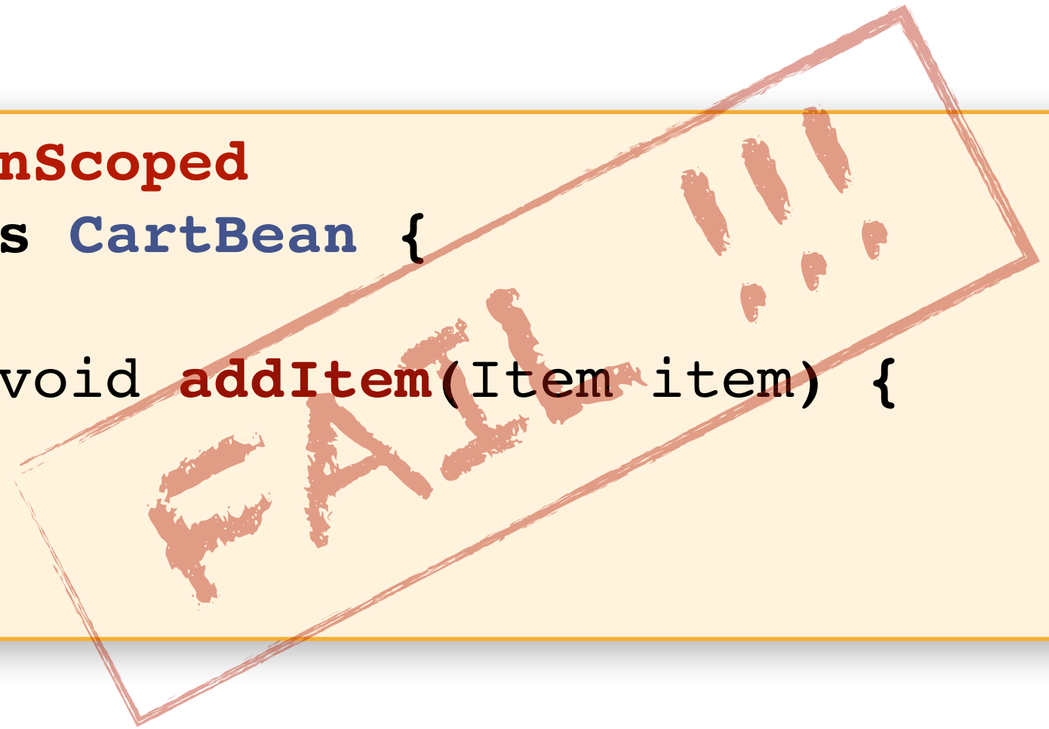
- Manage bean lifecycle
  - context helps container to choose when a bean should be generated and destroyed
  - it enforces the fact that a given bean is a singleton for a given context
- Built-in CDI contexts :
  - @Dependent (default)
  - @RequestScoped
  - @SessionScoped
  - @ConversationScoped
  - @ApplicationScoped
  - @Singleton
- You can create your own scope

Contexts

```
@SessionScoped
public class CartBean {

    public void addItem(Item item) {
        ...
    }
}
```

Contexts

```
@ApplicationScoped
public class CartBean {

    public void addItem(Item item) {
        ...
    }
}
```



Contexts

```
@ConversationScoped
public class CartBean {

    public void addItem(Item item) {
        ...
    }
}
```

Contexts

```
@ThreadScoped
public class CartBean {

    public void addItem(Item item) {
        ...
    }
}
```

Contexts

```
@HourScoped
public class CartBean {

    public void addItem(Item item) {
        ...
    }
}
```

Contexts

```
@RandomScoped
public class CartBean {

    public void addItem(Item item) {
        ...
    }
}
```

Producers

```
@Produces
public MyNonCDIClass myProducer() {
return new MyNonCdiClass();
}
...
@Inject
MyNonCDIClass bean;
```

Producers

```
@Produces
@RequestScoped
public FacesContext produceFacesContext() {
      return FacesContext.getCurrentInstance();
}
```

Producers

```
@Produces
public Logger produceLog(InjectionPoint injectionPoint) {
    return
Logger.getLogger(injectionPoint.getMember().getDeclaringClass().getName());
}
```

Decorators

```
@Decorator
@Priority(Interceptor.Priority.APPLICATION)
public class HelloDecorator implements HelloService {

    @Inject @Delegate HelloService service;

    public String hello() {
        return service.hello() + "-decorated";
    }
}
```

Interceptors

```
@InterceptorBinding
@Target({METHOD, TYPE})
@Retention(RUNTIME)
public @interface Loggable {}
```

Interceptors

```
@Interceptor @Loggable
public class LogInterceptor {
  @AroundInvoke
  public Object log(InvocationContext ic) throws Exception {
    System.out.println("Entering " + ic.getMethod().getName());
        try {
            return ic.proceed();
        } finally {
            System.out.println("Exiting " + ic.getMethod().getName());
        }
    }
}
```

Interceptors

```
@Loggable
public class MyBean {

    @Inject HelloService service;

    public void displayHello() {
        display( service.hello();
    }
}
```

Events

```
@Inject Event<Post> evt;

…

evt.fire(new Post("John Doe ", "Hello", ));

…

public void receiveEvt(@Observes Post evt) {
    System.out.println("Received : " + evt.message());
}
```

CDI extension

DEVOXX™

Extension for what use ?

- To change CDI container meta-data:
  - AnnotatedType
  - InjectionPoint / InjectionTarget
  - BeanAttributes/Beans
  - Producer
  - Observer

- By observing the events triggered by CDI at boot time

How to build a CDI extension

- Create a class implementing `javax.enterprise.inject.spi.Extension`
- Add some method that observes CDI lifecycle events to modify Bean Manager meta data
- Add file : `META-INF/services/javax.enterprise.inject.spi.Extension` containing extension classname

For Instance

- We can observe ProcessAnnotatedType event to change a type information or force the container to ignore it.

```
public interface ProcessAnnotatedType<X> {

    public AnnotatedType<X> getAnnotatedType();

    public void setAnnotatedType(AnnotatedType<X> type);

    public void veto();
}

public interface AnnotatedType<X> extends Annotated {

   public Class<X> getJavaClass();

   public Set<AnnotatedConstructor<X>> getConstructors();

   public Set<AnnotatedMethod<? super X>> getMethods();

   public Set<AnnotatedField<? super X>> getFields();
```

Example 1 : Ignoring JPA entities

- A commonly admitted good practice is to avoid using JPA entities as CDI beans.
- This extension excludes entities from the set of types discovered by CDI.

```
public class VetoEntity implements Extension {
    /**
     * Version CDI 1.0
     */
    public void vetoEntityLegacy(@Observes ProcessAnnotatedType<?> pat) {
       AnnotatedType at = pat.getAnnotatedType();
       if (at.isAnnotationPresent(Entity.class))
           pat.veto();
    }
    /**
     * Version CDI 1.1+
     */
    public void vetoEntity(@Observes @WithAnnotations({Entity.class})ProcessAnnotatedType<?> pat) {
       pat.veto();
    }
}
```

Things to know

- Once application is bootstrapped, the Bean Manager is in read only mode (no dynamic bean registration)
- Extensions are launched during bootstrap and are based on CDI events
- You only have to @Observes built-in CDI event to create your extensions

Example 2 : Add a Bean to CDI container

- There are many ways to add a bean to those automatically discovered at boot time
- The easier method is to add an AnnotatedType to those discovered by CDI.

Adding a HashMap Bean 1/2

```
public class HashMapAsBeanExtension implements Extension{

    public void addHashMapAsAnnotatedType(@Observes BeforeBeanDiscovery bbd,
                                          BeanManager beanManager)
    {
        bbd.addAnnotatedType(beanManager.createAnnotatedType(HashMap.class));
    }
}
```

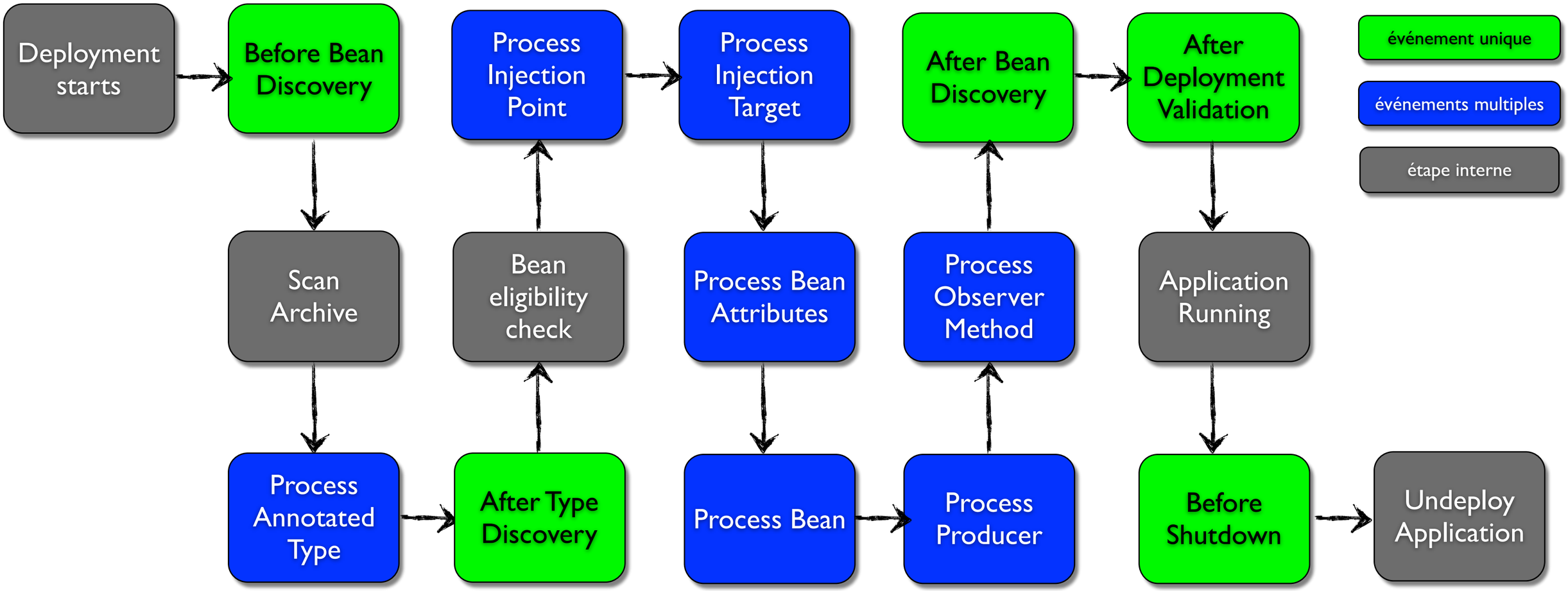
- Observing BeforeBeanDiscovery event we can add a new AnnotatedType based on HashMap classe.

Adding a HashMap Bean 2/2

- Once created we can add CDI service on this HashMap bean. See that Decorator.

```
@Decorator
@Priority(Interceptor.Priority.APPLICATION)
public abstract class MapDecorator implements Map{

    @Inject
    @Delegate
    Map delegate;

    @Override
    public Object put(Object key, Object value) {
        System.out.println("------- Putting Something in the Map ----------");
        if ("key".equals(key)) {
            System.out.println("==== Not adding key key ======");
            return null;
        }
        return delegate.put(key,value);
    }
}
```

CDI 1.1 Lifecycle



Scheduler Extension1/4

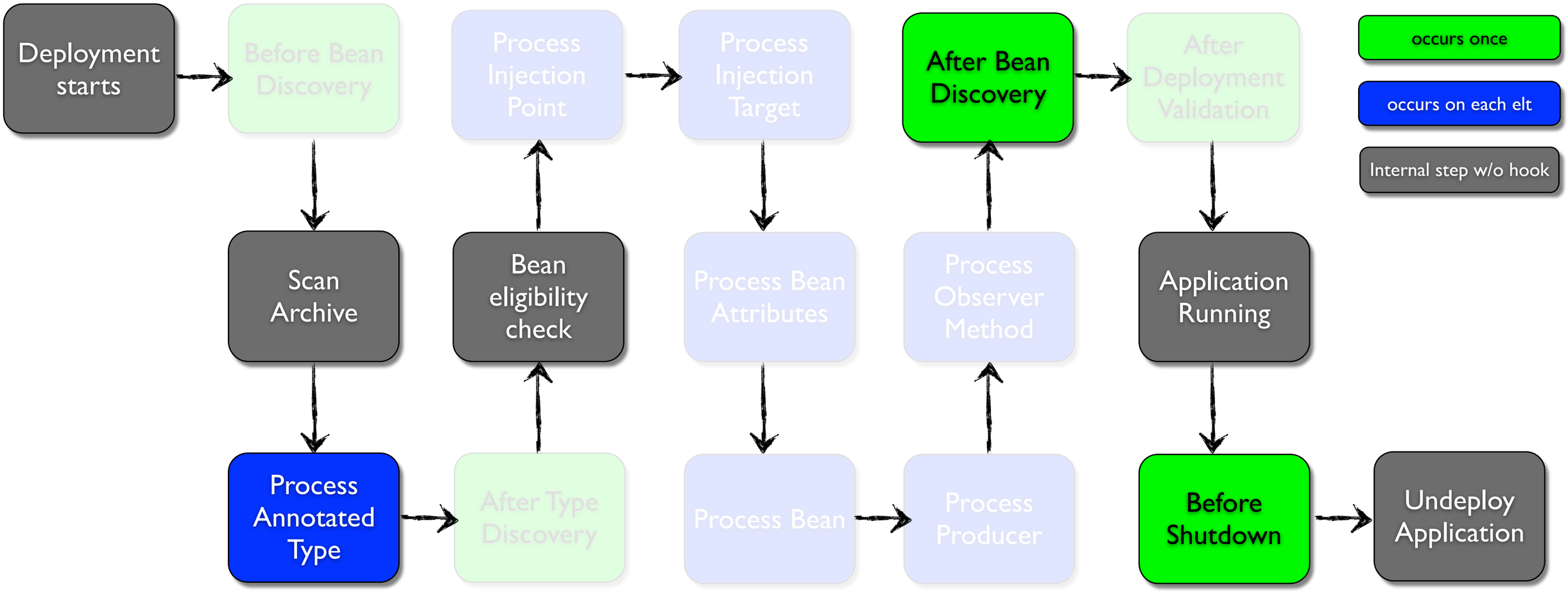
- This extension example comes from Apache Deltaspike project. It’s a simplified version.

- We want to be able to schedule jobs from by using annotation @Schedule.

Scheduler extension 2/4 (extension code)

```
public class SchedulerExtension implements Extension {
    private List<Class> foundManagedJobClasses = new ArrayList<Class>();
    private Scheduler scheduler;

    public <X> void findScheduledJobs(@Observes @WithAnnotations({ Scheduled.class }) ProcessAnnotatedType<X> pat) {
        Class<X> beanClass = pat.getAnnotatedType().getJavaClass();
        this.foundManagedJobClasses.add(beanClass);
    }

    public <X> void scheduleJobs(@Observes AfterBeanDiscovery afterBeanDiscovery, BeanManager beanManager) {
        initScheduler(afterBeanDiscovery)
        List<String> foundJobNames = new ArrayList<String>();
        for (Class jobClass : this.foundManagedJobClasses) {
            foundJobNames.add(jobClass.getSimpleName());
            this.scheduler.registerNewJob(jobClass);
        }
    }

    public <X> void stopScheduler(@Observes BeforeShutdown beforeShutdown) {
        if (this.scheduler != null) {
            this.scheduler.stop();
            this.scheduler = null;
        }
    }

    private void initScheduler(AfterBeanDiscovery afterBeanDiscovery) {
        this.scheduler = new QuartzScheduler();
        this.scheduler.start();
    }
}
```

Steps used in the extension



Scheduler Extension 3/4 (scheduler interface)

```
public interface Scheduler<T>
{
    void start();

    void stop();

    void pauseJob(Class<? extends T> jobClass);

    void resumeJob(Class<? extends T> jobClass);

    void interruptJob(Class<? extends T> jobClass);

    boolean isExecutingJob(Class<? extends T> jobClass);

    void registerNewJob(Class<? extends T> jobClass);

    void startJobManually(Class<? extends T> jobClass);

    <S> S unwrap(Class<? extends S> schedulerClass);
}
```

Scheduler extension 4/4 (scheduler producer)

```
@ApplicationScoped
public class SchedulerProducer
{
    @Inject
    private SchedulerExtension schedulerExtension;

    @Produces
    @ApplicationScoped
    protected Scheduler produceScheduler()
    {
        return this.schedulerExtension.getScheduler();
    }
}
```