Main sponsor

EJB 3.1 vs Contexts and Dependency Injection (CDI) and Dependency Injection for Java in Java EE 6

Jacek Laskowski

jacek@japila.pl

Jacek Laskowski

- Blogger of http://blog.japila.pl
- Blogger of http://jaceklaskowski.pl
- Member of Confitura 2011 conference organizing committee http://confitura.pl
- Twitters as @jaceklaskowski
- Committer of Apache OpenEJB and Apache Geronimo
- Works as a specialist for IBM WebSphere in IBM Poland

Java conference

Confitura 2011

http://confitura.pl

June, 11th

Warsaw, Poland

(formerly Javarsovia)

Let's begin with...

Enterprise JavaBeans 3.1

Enterprise JavaBeans is
a standard component model of Java, managed,
object-oriented, distributed, secure,
transactional business applications

```
@Stateless
public class HelloInEnglish implements Hello {

    @Override
    public String sayHello(String whom) {
        return "Hello, " + whom;
    }

}
```

```
@Stateful
public class HelloInEnglish {

    public String sayHello(String whom) {
        return "Hello, " + whom;
    }

}
```

```
@MessageDriven
public class HelloInEnglish {

    public void onMessage(Message message) {
        // …
    }

}
```

```
@MessageDriven(
    activationConfig =  {
        @ActivationConfigProperty(
            propertyName = "destinationType",
            propertyValue = "javax.jms.Queue")
    })
public class HelloInEnglish implements MessageListener {

    public void onMessage(Message message) {
        // …
    }

}
```

Enterprise JavaBeans is NOT
a silver bullet for every business application

```
@Stateless
public class TestFacadeEJB implements Hello {

    @EJB(beanName="HelloInEnglish")
    Hello hello;

    @Override
    public String sayHello(String whom) {
        return hello.sayHello(whom);
    }

}
```

Enterprise JavaBeans is NOT
type-safe dependency injection framework

Welcome to...
Dependency Injection for Java (JSR-330)

Dependency Injection for Java is
type-safe dependency injection framework

The types on which a type depends are known as its dependencies. The process of finding an instance of a dependency to use at run time is known as resolving the dependency.

Package javax.inject Description

javax.inject package

- a single interface - javax.inject.Provider<T>
- @Inject marks a place to inject
- @Qualifier identifies new qualifier annotations
- @Named for stringified @Qualifiers
- @Scope identifies scope annotations
- @Singleton is a @Scope for once-instantiated types

javax.inject vs javax.ejb

- @Singleton is a @Scope for once-instantiated types
- @Singleton is a singleton session bean

```
@Singleton
@Lock(READ)
public class ComponentRegistryBean implements ComponentRegistry {

    private final Map<Class, Object> components = new HashMap<Class, Object>();

    public <T> T getComponent(Class<T> type) {
        return (T) components.get(type);
    }

    @Lock(WRITE)
    public <T> T setComponent(Class<T> type, T value) {
        return (T) components.put(type, value);
    }

    @Lock(WRITE)
    public <T> T removeComponent(Class<T> type) {
        return (T) components.remove(type);
    }

}
```

openejb3/examples/simple-singleton

```
@Singleton
public class JavaLoggerFactory implements LoggerFactory {

    public <T> T getLogger(Class<?> clazz, Class<T> type) {
        Object logger = Logger.getLogger(clazz.getName());
        return type.cast(logger);
    }
}
```

openwebbeans/samples/standalone-sample

EJB vs DI4j

- @Resource
- @EJB
- @PersistenceContext
- @PersistenceUnit
- @WebServiceRef

- @Inject

All rise....

Contexts and Dependency Injection for the Java EE Platform (JSR-229)

...is now in session

CDI is
a set of services for object lifecycle, dependency injection, Unified EL support, decoration, interceptors, event notification and extensions.

CDI defines a bean
that *is a source of contextual objects which define application state and/or logic.*

JSR-299: Contexts and Dependency Injection for the Java EE platform Chapter 2. Concepts

```
@Stateless
public class HelloInEnglish implements Hello {

    @Override
    public String sayHello(String whom) {
        return "Hello, " + whom;
    }

}
```

```
public class HelloInEnglish implements Hello {

    @Override
    public String sayHello(String whom) {
        return "Hello, " + whom;
    }

}
```

```
@Named(“hello”)
public class HelloInEnglish implements Hello {

    @Override
    public String sayHello(String whom) {
        return "Hello, " + whom;
    }

}
```

```
@ManagedBean
public class HelloInEnglish implements Hello {

    @Override
    public String sayHello(String whom) {
        return "Hello, " + whom;
    }

}
```

```
public class HelloInEnglish implements Hello {

    HelloInEnglish(String whom) {
        // …
    }

    @Override
    public String sayHello(String whom) {
        return "Hello, " + whom;
    }

}
```

```
@MessageDriven
public class HelloInEnglish implements MessageListener {

    public void onMessage(Message message) {
        // …
    }

}
```

EJB vs CDI

- @MessageDriven in EJB
- @Observes + @Inject Event or BeanManager

```
@MessageDriven
public class HelloInEnglish implements MessageListener {

    public void onMessage(Message message) {
        // …
    }

}
```

```
import javax.enterprise.event.Observes;

public class EnabledBean {
    public void observerMethod(@Observes String firstName) {
        System.out.println("Observed firstName: " + firstName);
    }
}
```

```
@Inject
BeanManager manager;

@Inject
Event<String> firstNameEvent;

firstNameEvent.fire("Jacek");
manager.fireEvent("Agatka", new Annotation[0]);
```

Thanks!