Interceptor and Decorator CDI workshop doc

Warning

This document is a proposal in draft state. Its purpose is to be a starting point for discussion on new features or enhancement.

Contribution to this document

Contributions to this document are open the the community, but they can't be done anonymously. If you want to comment or suggest modification / new content, please make sure you are connected with your google Account and ask access by clicking on the share button at the upper right corner of the screen.

By asking the right to contribute you agree that:

- For all code provided, you are licensing the code under the Apache License, Version 2.
- For all other ideas provided, the provider waives all patent and other intellectual property rights inherent in such information.

Introduction

Current Discussions

Which methods get intercepted?

The Interceptor-1.2 spec defines that the following methods might get intercepted:

- Business Methods
- Lifecycle Event Callbacks (PostConstruct and PreDestroy
- Constructors (new in interceptors-1.2)

What is a 'Business Method'?

The definition of what a 'Business Method' is can be found in the EJB specification. It's not yet clear if this definition is also binding for the interceptors spec itself and thus also for CDI.

What is a ‘Lifecycle Event Callback Interceptor’?

What is a ‘Constructror Interceptor’?

Which methods need to be proxied?

Are final methods allowed on classes which need to get proxied?

This is different between EJB and CDI. CDI doesn’t allow them whereas EJB does.

How to call private methods on the proxied instance?

‘Unwrapping’ proxies to get the internal instance is mostly interesting for e.g. delivering CDI events to private observer methods. It is also important for some frameworks who need to store info in Maps or Sets and thus need to access the ‘internal’ instance for equals() and hashCode().

Mirror Annotations to our proxies

Currently the spec says nothing about how annotations on the original classes get handled if a proxy gets applied. If the annotations do not get ‘copied’ over to the subclass proxy-class as well then frameworks like EL, etc could not evaluate those annotations by just interspecting the class of the object they get.
We should define that all the annotations of the proxied classes shall get mirrored on the generated subclass proxy classes.