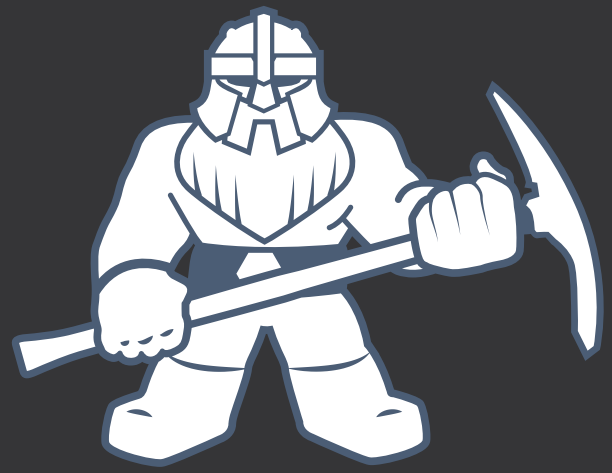
Java CDI

O que é, como se reproduzem e onde vivem os componentes automágicos



Usando Java CDI em projetos Jakarta EE ou Microprofile.io

Este quem vos fala...

Victor Osório
Senior Software Engineer @ Openet

Mais de 14 anos de experiência com desenvolvimento Java

14 de Java SE!
02 de ~~Java EE!~~ Jakarta EE!

https://twitter.com/vepo

https://www.linkedin.com/in/victorosorio

https://github.com/vepo

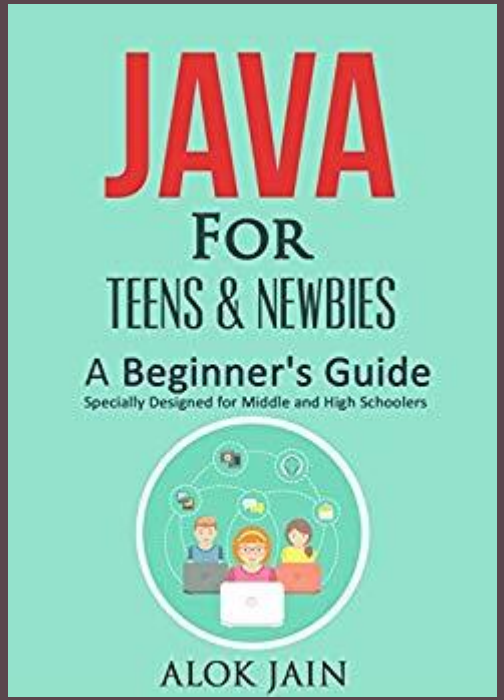
OPENET

Agenda

- O que é Java CDI
  - Porque Usar
  - Jakarta EE
  - Microprofile.io
- Definições
- Criando Projetos
- Casos de Uso
  - Como declarar
  - Como usar
  - Interceptando Chamadas
  - Decorators
  - Eventos
- Java CDI em Java SE

Público Alvo

- Desenvolvedor Java
- Interesse em Jakarta EE
- Interesse em Desenvolvimento Web
- Interesse em melhorar qualidade do Código Desenvolvido



Java CDI: O que é?

- Contexts and Dependency Injection (http://www.cdi-spec.org/)
- Apenas uma Especificação
  - Não é uma biblioteca
- Base das outras Especificações
- Permite
  - Programação Orientada a Aspecto
  - Inversão de Controle
  - Controle de Contexto
  - TypeSafe!

Porque usar Java CDI

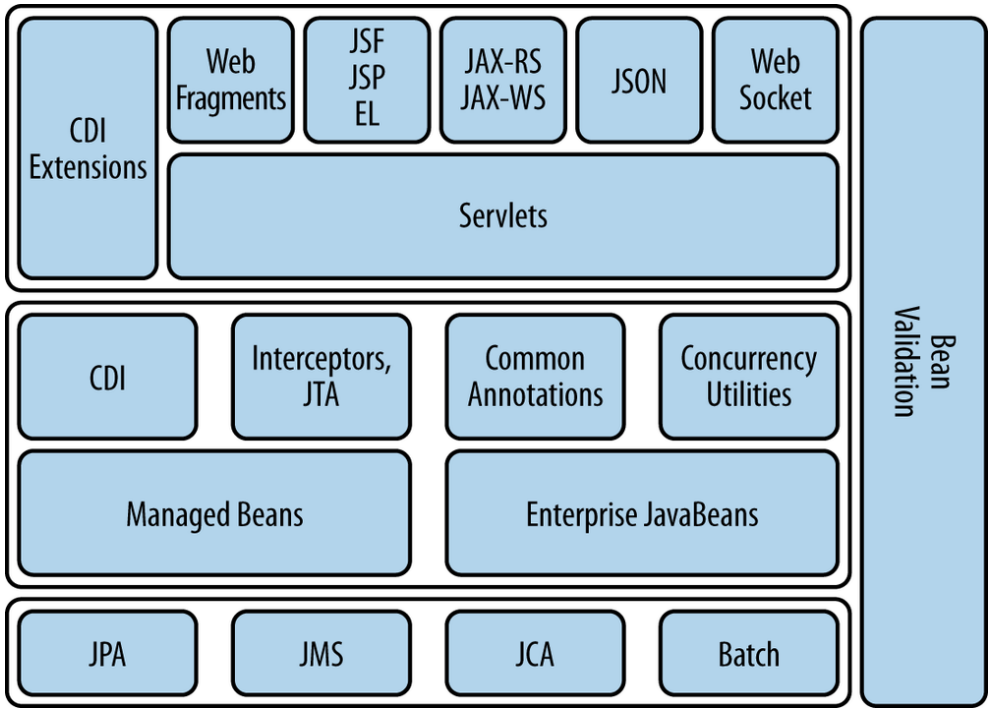
- Velocidade de Desenvolvimento
- Padronização do código
- Baixo acoplamento e alta coesão
- Foco na Lógica de Negócio
- Jakarta EE e Microprofile.io

O que é Jakarta EE?

- Modelo de Programação
  - Aprenda 1 Framework!!!!!
  - Orientado a Padronização
  - Não dependente de um Framework!
  - Você não precisa fazer o parser da Requisição HTTP para criar um servidor HTTP

- ~~Dar um passo atrás para dar dois a frente!~~

- Há um conjunto enorme de especificações:
  - https://stackoverflow.com/questions/37082364/a-summary-of-all-java-ee-specifications



O que é Microprofile.io

- Conjunto de Especificações paralelo ao Jakarta EE
- Pontapé inicial para desenvolvimento de microserviços em Java
- Padrões de https://microservices.io

Definições

Vamos definir algumas coisas antes de iniciarmos...

Container (CDI)

- No mundo CDI, container é onde residem os Beans.
  - É quem cria os Beans
  - Quem os alimenta
  - E quem os elimina!
- É a instância do Servidor, não é o Container Docker
  - Mas é o que podemos chamar de Servidor de Aplicação.

Inversão de Controle

- É o que difere Bibliotecas de Frameworks – Martin Fowler
- Framework é responsável por inicializar e configurar componentes
- Também se refere Eventos (Consumers e Producers)
- Exemplos:
  - *Eu não quero saber como Instanciar um Cliente de Banco de Dados*
  - *Eu não quero criar Transações de Banco de Dados*
  - *Eu não quero ...*
- A responsabilidade por gerenciar isso é delegada! Quem usa não precisa saber.

Programação Orientada a Aspecto

- Refere-se a separação de Conceitos: o que é o principal e o secundário?
- O Código será para o que é principal: A lógica de Negócios
- Os aspectos do código serão definidos “fora” do código
  - Transações
  - Log
  - Etc...
- Melhor modularização
- Melhor visibilidade da lógica de negócios

EJB

- É uma especificação muito mais complexa que o CDI
- EJB pode ser:
  - Transactional
  - Remote or local
  - Able to passivate stateful beans freeing up resources
  - Able to make use of timers
  - Can be asynchronous

Não vamos falar de EJB!!!! Tá ok?

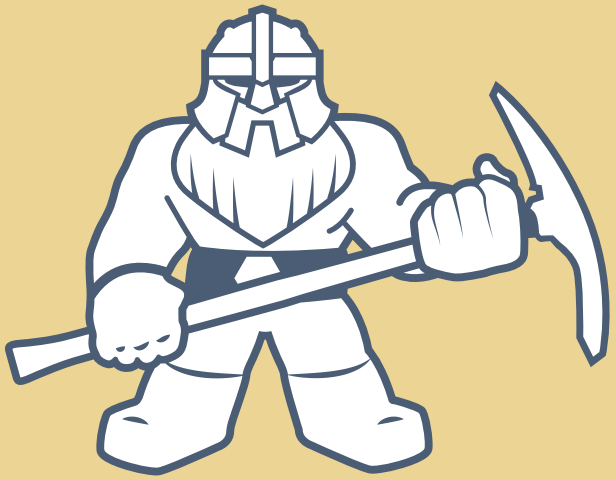
Java CDI

Explicação por usos...

Criando um Projeto CDI

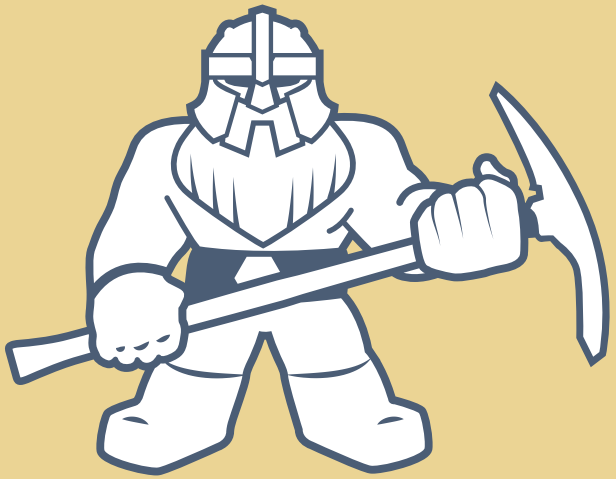
- Use um starter: https://start.microprofile.io/
- Adicionar o arquivo `src/main/resources/META-INF/beans.xml` ao projeto Maven

```
<beans xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/beans_1_1.xsd"
    bean-discovery-mode="all">
</beans>
```

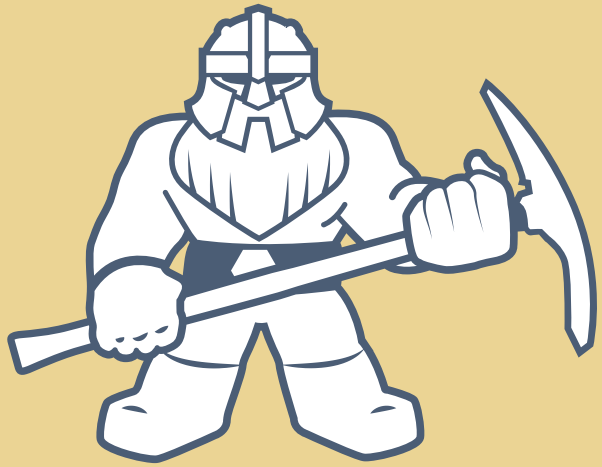


Instanciação

- Qualquer Bean pode ser Injetado em outro Bean
- Requisitos
  - Não é uma Inner Class
  - Não é uma Classe abstrata e não possui a Annotation `@Decorator`
  - Não implementa a interface `javax.enterprise.inject.spi.Extension`
  - Não possui a Annotation `@Vetoed` e nem está em um pacote com `@Vetoed`
  - Tem um construtor apropriado
    - Um construtor sem parâmetros
    - ~~Um construtor com parâmetros com `@Inject`~~
      - Está na especificação, mas não funciona! 🤔 🤦‍♂️



Contexto e Escopo

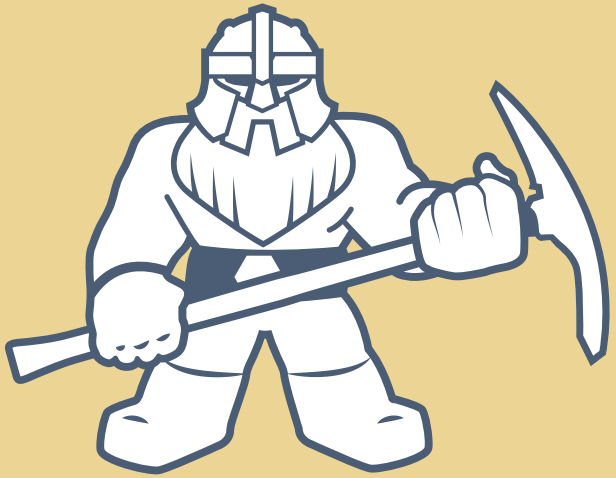


- CDI tem suporte a Escopos e Contextos. Ver Documentação

Scope	Annotation	Duration
Request	@RequestScoped	A user's interaction with a web application in a single HTTP request.
Session	@SessionScoped	A user's interaction with a web application across multiple HTTP requests.
Application	@ApplicationScoped	Shared state across all users' interactions with a web application.
Dependent	@Dependent	The default scope if none is specified; it means that an object exists to serve exactly one client (bean) and has the same lifecycle as that client (bean).
Conversation	@ConversationScoped	A user's interaction with a JavaServer Faces application, within explicit developer-controlled boundaries that extend the scope across multiple invocations of the JavaServer Faces lifecycle. All long-running conversations are scoped to a particular HTTP servlet session and may not cross session boundaries.

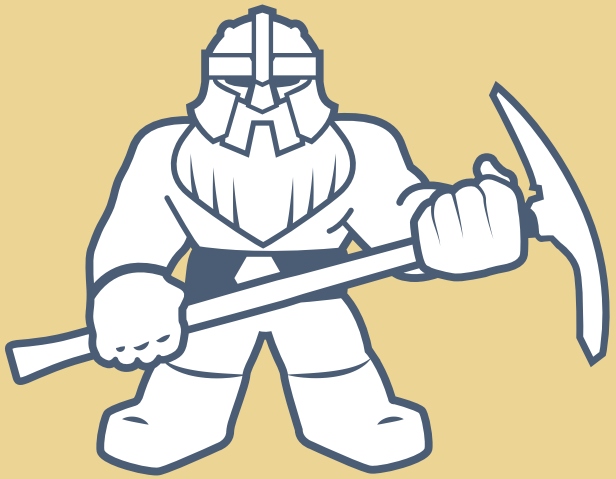
Instanciando

- Diretamente
  - Ver UserRepository
  - Qualquer classe pode ser carregada como Bean
  - Usar @Inject
  - O Escopo deve ser definido na classe a ser instanciada



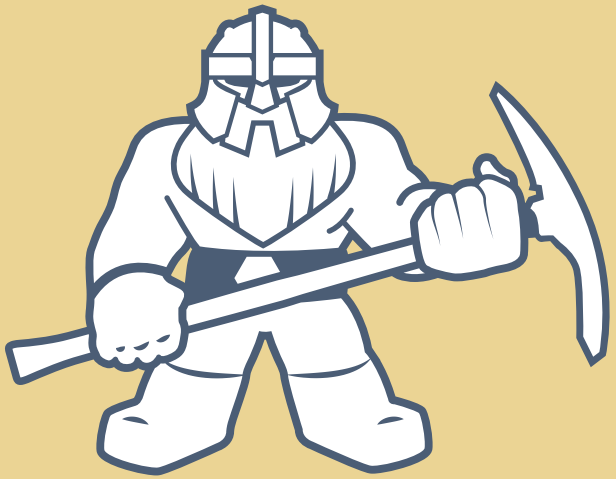
Instanciando – Ciclo de Vida

- Uso do @PostConstruct e @PreDestroy
  - Ver MongoClientFactory
  - Beans aceita definir Métodos para Instanciação e Finalização
  - Metodos são chamados pelo Container, no momento que o Container decidir.



Instanciando – Factory

- Via @Produces
  - Ver MongoClientFactory
  - É possível criar classe de Factories
    - A vantagem é não duplicar recursos quando necessário.



Injetando

- Simples Inject
  - Ver UserEndpoint
  - Deve ser usado em um Bean!
  - Não é necessário saber o escopo do Bean
  - Não é necessário saber nada sobre o Bean
    - Exceto se ele usa Qualifiers ou Named *a seguir*...

Injetando – Especialização

- Java CDI aceita Especialização
  - A classe que usará o Bean não precisa saber a implementação
- Por Qualifier
  - Ver HelloEndpoint
  - Definir Annotation com Qualifier
- Por Naming
  - Ver ByeEndpoint
  - Não é necessário Annotation com Qualifier, mas @Named

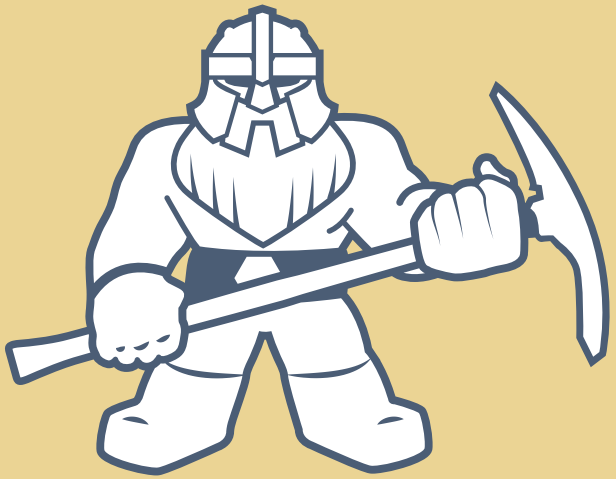
Interceptando

- O Bean injetado nem sempre é exatamente uma implementação direta de sua definição
  - Lógicas acessórias podem ser adicionadas
- Quando usar?
  - Quando o que deve ser feito não é parte da Lógica de Negócios direta
- Configurando...
  - Deve ser declarado no beans.xml
  - Deve usar Qualifier
  - Deve declarar um Interceptor
  - Ver UserEndpoint

Decorators

- Centralização de Lógica de Negócios
- Muito similar ao *Interceptors*, mas TypeSafe
- É possível adicionar “decoradores” em todas as implementações de um mesmo Bean.

- Ver HelloServiceValidator



Eventos CDI

- Java CDI provê uma interface para Ouvir/Produzir eventos
  - Eventos a nível de JVM
  - Tipo de evento: Qualquer classe Java
  - Produtor: classe Event
  - Consumidor: Método com @Observes
    - Aceita @Priority (CDI 2.0)
- Exemplos
  - Producer
    - Ver UserEndpoint
  - Consumer
    - Ver Consumers

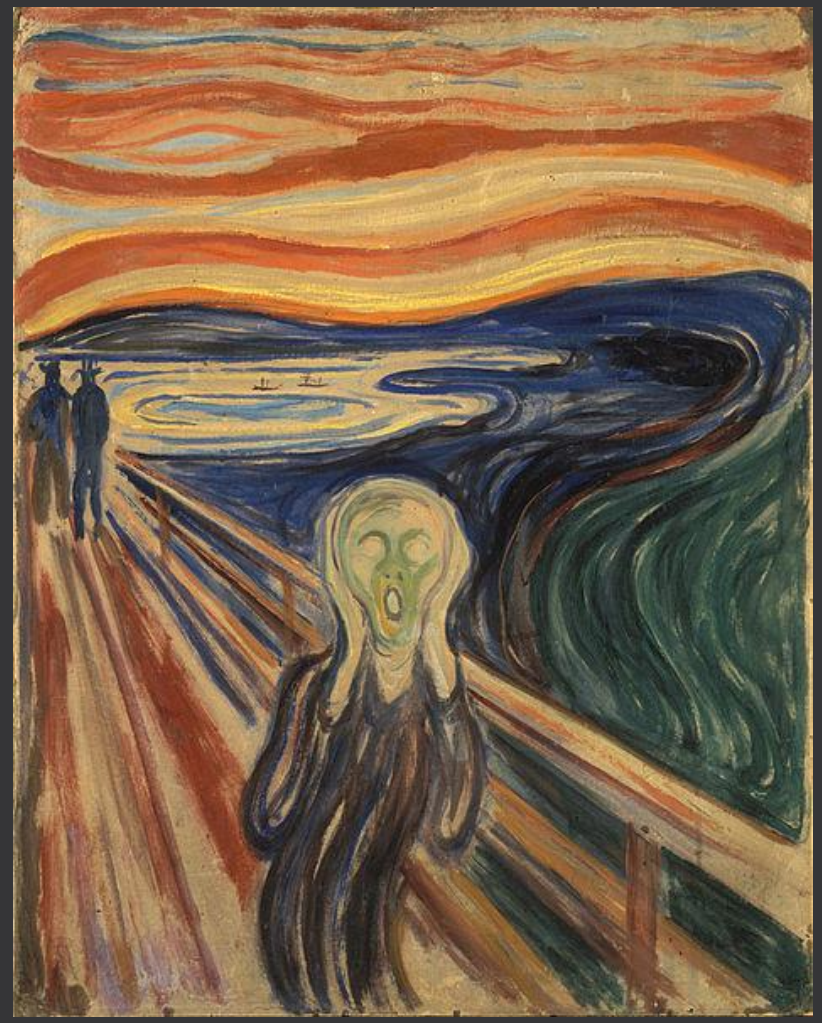
Java CDI em Java SE

- Usar Weld
  - https://weld.cdi-spec.org/
- Deve adicionar beans.xml
- Dependências
  - Não deve declarar dependências como “provided”
  - Deve gerar um “fat jar”
- Deve declarar um main

```
public static void main(String[] args) {
	SeContainerInitializer initializer = SeContainerInitializer.newInstance();
	try (SeContainer container = initializer.initialize()) {
		CdiRunner runner = container.select(CdiRunner.class).get();
		runner.run();
	}
}
```

Duvidas?

Não existe pergunta boba!



Obrigado!!!

- Códigos
  - https://github.com/vepo/cdi-tutorial
  - https://github.com/vepo/cdi-tutorial-java-se

- Dicas
  - JakartaOne Livestream
    - https://www.youtube.com/playlist?list=PLy7t4z5SYNaSxBfGMW-NRQkV_qWM0NipB
    - Live Code de Microprofile.io e JakartaEE