Formation Java EE

CDI

digitalent FORMATION

CDI ?

Context and Dependency Injection

- Couplage faible avec un typage fort
- Injection de dépendance avec types
- Intercepteurs
- Notifications d'événements
- Extensions

Implémentation de référence : Weld

CDI injecte des "beans"

Bean ?

Un objet issu d'une classe qui a au moins un des éléments suivants :

- un constructeur sans paramètres
- un constructeur annoté avec l'annotation @Inject
- un producteur de beans

Bean ?

Activer CDI

- Ajouter un fichier vide “beans.xml” dans le répertoire WEB-INF pour les WAR
- Ajouter un fichier vide “beans.xml” dans le répertoire META-INF pour les JAR

Exemple d'injection

```
public class PizzaService {

    public List<Pizza> findAllPizze(){
        ...
    }
}
```

Injection par type
Qualifier = @Default

```
@WebServlet("/pizze")
public class PizzaWeb extends HttpServlet {
    Plus besoin de new....  il utlise des interfaces.
    @Inject private PizzaService pizzaService;
```

Gestion des ambiguïtés

```
public class CommandeService {}

public class CommandeEntrepriseService
                extends CommandeService {}

public class CommandeParticulierService
                extends CommandeService {}

public class PizzaService {

    @Inject CommandeService commandeService;
```

QUEL TYPE INJECTE ?

CommandeService ?
CommandeEntrepriseService ?
CommandeParticulierService ?

Gestion des ambiguïtés avec @Qualifier

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

public class CommandeService {}
```

Singleton simple --> au lieu de coder avec new... qui fait plusieurs instances

```
@Entreprise
public class CommandeEntrepriseService extends CommandeService {}

public class CommandeParticulierService extends CommandeService {}

public class PizzaService {

    @Inject @Entreprise CommandeService service;
```

Gestion des ambiguïtés avec @Named

```
public class CommandeService {}

@Named
public class CommandeEntrepriseService extends CommandeService {}

public class CommandeParticulierService extends CommandeService {}

public class PizzaService {

    @Inject @Named("commandeEntrepriseService") CommandeService
service;
```

@PostConstruct, @PreDestroy

```
@Named
public class CommandeEntrepriseService extends CommandeService {
    @Inject private PizzaServices pService;
     @PostConstruct
     public void onPostConstruct() {
          ...
     }

     @PreDestroy
     public void onPreDestroy() {
          ...
     }

}
```

"Donc ils associé -- Commande service & PizzaSevices"

exemple : pour rajouter des services à injecter

Par exemple : initialiser une liste de pizzas

pService...

Scope d'un bean CDI

La durée de vie d'un bean

@RequestScoped

- Le bean existe le temps d'une requête HTTP

@SessionScoped

- Le bean existe le temps d'une session serveur

@ApplicationScoped --> Pour faire un singleton

- Le bean existe tout au long de la vie de l'application

@Dependent

- Scope par défaut. Le bean prend le même cycle de vie que le bean dans lequel il est injecté

@ConversationScoped

- Le bean vit le temps d'une conversation dans une application JSF.

Intercepteurs

```
public class LogInterceptor {

	@AroundInvoke
	public Object log(InvocationContext context) throws Exception {

		// traitement avant l'invocation de la méthode

		Object b =  context.proceed();

		// traitement après l'invocation de la méthode

		return  b;
	}
}

@Interceptors(LogInterceptor.class)
public class PizzaService {
```

Ca veut dire exécuter la vraie méthode

attention a ne pas mettre NULL--> en debugger il va bien marcher mais ensuite va supprimer

Producteur de beans

@Produces

- Permet de faire participer à l'injection de dépendance des objets créés à partir de classe "non CDI".
- Création via une méthode d'instance

```
public class Utils {

  @Produces
  public ConfigApp getConfig() {
    return new ConfigApp();
  }

}
```

Producteur de beans

@Produces

- Création via une méthode statique

```
public class Utils {

  @Produces
  public static ConfigApp getConfig() {
    return new ConfigApp();
  }

}
```

Producteur de beans

@Produces

- Création via un attribut d'instance

```
public class Utils {

  @Produces private ConfigApp configApp = new ConfigApp();

}
```

Producteur de beans

@Produces

- Création via un attribut statique

```
public class Utils {

  @Produces private static ConfigApp configApp = new ConfigApp();

}
```

Producteur de beans

@Produces

- Le scope est configurable

```
public class Utils {

  @Produces
 @ApplicationScoped
  public ConfigApp getConfig() {
    return new ConfigApp();
  }

}
```

Producteur de beans

@Produces

- Le nom du bean est configurable

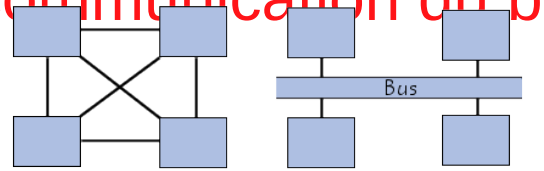
```
public class Utils {

  @Produces
 @Named("configApp")
  public ConfigApp getConfig() {
    return new ConfigApp();
  }

}
```

Evénements

Exemple de Rossi SNCF --> retard de train... communication du bus



javax.enterprise.event.Event et @Observes

- CDI offre un bus d'événements permettant un découplage entre différents services
- Un événement est un objet POJO classique
- L'interface Event est utilisée pour émettre un événement

```
@Inject private Event<MonEvent> monEvent;
```

- L'annotation @Observes permet d'écouter un événement

```
public void ecouteMonEvent(@Observes MonEvent event) {
    …
}
```