11

Mãos a obra: TypedQuery

Na classe `UsuarioDao` estamos criando uma query, hum... Vamos usar o **CDI** para produzir nossa query.

Como `TypedQuery` é uma interface teremos que produzir esse objeto. No projeto `Alura-Lib` vamos criar uma classe que produz `TypedQuery` s.

```
public class TypedQueryFactory {

    @Inject
    private EntityManager em;

    @Produces
    public <X> TypedQuery<X> factory(InjectionPoint point){

        ParameterizedType type = (ParameterizedType) point.getType();

        @SuppressWarnings("unchecked")
        Class<X> classe = (Class<X>) type.getActualTypeArguments()[0];

        String jpql;
        return em.createQuery(jpql, classe);
    }

}
```

Agora que temos nossa classe `TypedQueryFactory` pronta basta resolver o problema da `JPQL` que queremos utilizar para criar um `TypedQuery` . Como `JPQL` é algo que distingue um `TypedQuery` de outro. Vamos criar um qualificador que recebe uma `string` .

```
@Qualifier
@Target({ElementType.PARAMETER, ElementType.FIELD, ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
public @interface Query {
    String value();
}
```

Agora temos um qualificador `@Query` que utiliza uma `string` para qualificar. Perceba que dessa forma teremos que criar um produtor para cada variação dessa `string` o que seria inviável.

E se pararmos para pensar a `string` é uma informação que queremos utilizar dentro do nosso produtor e não algo para diferenciar uma query de outra. Não queremos que o **CDI** considere essa `string` para qualificar nossas querys.

Então precisamos informar ao **CDI** que ele não deve utilizar essa `string` para qualificar nossas querys. Para isso na anotação `@Query` podemos anotar o método `String value()` com `@Nonbinding` .

```
@Qualifier
@Target({ElementType.PARAMETER, ElementType.FIELD, ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
```

```
public @interface Query {
    @Nonbinding String value();
}
```

Voltando agora à nossa classe `TypedQueryFactory` vamos adicionar esse qualificador e pegar nossa `JPQL` .

```
public class TypedQueryFactory {

    @Inject
    private EntityManager em;

    @Produces
    @Query("")
    public <X> TypedQuery<X> factory(InjectionPoint point){

        ParameterizedType type = (ParameterizedType) point.getType();

        @SuppressWarnings("unchecked")
        Class<X> classe = (Class<X>) type.getActualTypeArguments()[0];

        String jpql = point.getAnnotated().getAnnotation(Query.class).value();
        return em.createQuery(jpql, classe);
    }

}
```

Vamos instalar nosso `jar` no repositório local: Clique com o botão direito do mouse sobre o projeto `Alura-lib` e selecione `Run as > Maven Install` .

Agora basta atualizar o projeto `Livraria` : Clique sobre o projeto `Livraria` e selecione `Maven > Update project...` .

Pronto agora podemos receber nossa query injetada na nossa classe `UsuarioDao` .

```
public class UsuarioDao {


    @Inject @Query("select u from Usuario u where u.email = :pEmail and u.senha = :pSenha")
    private TypedQuery<Usuario> query;

    public boolean existe(Usuario usuario) {

        query.setParameter("pEmail", usuario.getEmail());
        query.setParameter("pSenha", usuario.getSenha());
        try {
            query.getSingleResult();
        } catch (NoResultException ex) {
            return false;
        }


        return true;
    }

}
```

Rode novamente o projeto `Livraria` e verifique se tudo continua funcionando.