
INNOVA-PATIKA

Hafta-2 Ödevi

Hatice Öztürk

18/01/2022

Inversion Of Control

Inversion of Control, Türkçe'ye kontrolün/bağımlılığın tersine çevrilmesi olarak da geçebilmektedir. Bu kavramı genellikle nesne yönelimli programlamada sıklıkla görmekteyiz.

Sınıfların ya da servislerin; bağımlılıklarının, yaşam döngüsünün (oluşturulmasının ve yok edilmesinin) kontrollerinin framework ya da bir container içerisine verilmesi işlemidir.

MySQL.java ×

```
1 package com.innova.databaseHomework.
2
3 public class Mysql implements Databa
4
5     @Override
6     public void databaseName() {
7         System.out.println("MySQL");
8     }
9
10 }
11
```

Postgresql.java ×

```
1 package com.innova.databaseHomework.
2
3 public class Postgresql implements D
4
5     @Override
6     public void databaseName() {
7         System.out.println("Postgres
8     }
9
10 }
11
```

Database.java ×

```
1 package com.innova.databaseHomework.
2
3 public interface Database {
4     public void databaseName();
5 }
6
```

Test.java ×

```
1 package com.innova.databaseHomework.IOCwithoutCDI;
2
3 public class Test {
4
5
6     public static void main(String[] args) {
7         DatabaseManager manager=new DatabaseManager(new Pos
8         manager.getDatabaseName();
9     }
10 }
11
```

DatabaseManager.java ×

```
1 package com.innova.databaseHomework.IOCwithoutCDI;
2
3 public class DatabaseManager {
4
5     Database databaseInterface;
6
7     public DatabaseManager(Database databaseInterface) {
8         this.databaseInterface = databaseInterface;
9     }
10     public void getDatabaseName() {
11         databaseInterface.databaseName();
12     }
13 }
14
```

CDI

Bir uygulama geliştirirken componentler arasındaki bağımlılık olmaması, uygulamanın karmaşıklığını önler. Böylece bağımsız componentler ile geliştirme sağlanarak uygulama daha sade bir hal alır. Bu aşamada CDI devreye girer. CDI köprü görevi görerek bu componentlerin haberleşmesini sağlar.

Java EE kullanırken APIler arasındaki bağımlılıklar sizin yerinize CDI ile yönetilir. Typesafe dependency injection sayesinde compiler zamanında bu bağımlılıklar ayarlanmış olur.

Alternative

*Mysql.java X

```
1 package com.innova.databaseHomework.alternative;
2
3 import javax.enterprise.inject.Alternative;
4
5 @Alternative
6 public class Mysql implements IDatabase{
7
8     @Override
9     public String databaseName() {
10         // TODO Auto-generated method stub
11         return "MySQL";
12     }
13
14 }
15
```

*Postgresql.java X

```
1 package com.innova.databaseHomework.alternative;
2
3 import javax.enterprise.inject.Alternative;
4
5 @Alternative
6 public class Postgresql implements IDatabase{
7
8     @Override
9     public String databaseName() {
10         // TODO Auto-generated method stub
11         return "PostgreSQL";
12     }
13
14 }
15
```

IDatabase.java X

```
1 package com.innova.databaseHomework.alternative;
2
3 public interface IDatabase {
4
5     public String databaseName();
6 }
7
```

DatabaseManager.java X

```
1 package com.innova.databaseHomework.alternative;
2
3 import javax.enterprise.context.ApplicationScoped;
4
5 @Named(value = "alternativeCDI")
6 @ApplicationScoped
7 public class DatabaseManager {
8
9     @Inject
10     IDatabase databaseInterface;
11
12     public String getDatabaseName() {
13         return databaseInterface.databaseName();
14     }
15
16 }
17
```

Qualifier

Uygulama içerisinde bir arayüzü uygulayan (implementation) birden fazla sınıf olabilir, bu arayüzün çağırımı sırasında bunu uygulayan hangi sınıfın gelmesi gerektiğini belirlemek karmaşıklığa neden olabilmektedir.

Postgresql.java ×

```
3 public class Postgresql implements IDatabase{
4
5     @Override
6     public String databaseName() {
7         // TODO Auto-generated method stub
8         return "PostgreSQL";
9     }
10
```

IDatabase.java ×

```
1 package com.innova.databaseHomework.qualifier;
2
3 public interface IDatabase {
4
5     public String databaseName();
6 }
```

QualifierDatabaseManager.java ×

```
1 package com.innova.databaseHomework.qualifier;
2
3 import java.io.Serializable;
4
5 @Named(value = "qualifierCDI")
6 @ApplicationScoped
7
8 public class QualifierDatabaseManager implements Serializable {
9
10     @Inject
11     @QualifierCokluSecim
12     private IDatabase databaseInterface;
13
14     public String getDatabaseName() {
15         return databaseInterface.databaseName();
16     }
17 }
18
19
20
```

Mysql.java ×

```
1 package com.innova.databaseHomework.qualifier;
2
3 import javax.enterprise.inject.Default;
4
5 @Default
6 public class Mysql implements IDatabase{
7
8     @Override
9     public String databaseName() {
10         // TODO Auto-generated method stub
11         return "MySQL";
12     }
13
14 }
15
```

QualifierCokluSecim.java ×

```
1 package com.innova.databaseHomework.qualifier;
2
3 import static java.lang.annotation.ElementType.FIELD;
4
5
6 @Qualifier
7 @Target({TYPE,METHOD,PARAMETER,FIELD})
8 @Retention(RUNTIME)
9 @Documented
10 public @interface QualifierCokluSecim {
11
12 }
13
14
15
```


EnumQualifier

IDatabaseName.java

```
1 package com.innova.databaseHomework.enumqualifier;
2
3 public interface IDatabaseName {
4
5     public String databaseName();
6 }
7
```

IDatabase.java

```
1 package com.innova.databaseHomework.enumqualifier;
2
3 public interface IDatabase {
4     public String databaseName();
5 }
6
```

Postgresql.java

```
1 package com.innova.databaseHomework.enumqualifier;
2
3 @EQualifier(EDatabaseName.POSTGRESQL)
4 public class Postgresql implements IDatabaseName{
5
6     @Override
7     public String databaseName() {
8         // TODO Auto-generated method stub
9         return "PostgreSQL";
10    }
11
12 }
13
```

EDatabaseName.java

```
1 package com.innova.databaseHomework.enumqualifier;
2
3 public enum EDatabaseName {
4
5     MySQL, POSTGRESQL, MONGODB
6 }
7
```

EQualifier.java

```
1 package com.innova.databaseHomework.enumqualifier;
2 import static java.lang.annotation.ElementType.FIELD;
3
4 @Qualifier
5 @Target({ TYPE, METHOD, PARAMETER, FIELD })
6 @Retention(RUNTIME)
7 @Documented
8 public @interface EQualifier {
9     EDatabaseName value();
10 }
11
```

QualifierManager.java

```
1 package com.innova.databaseHomework.enumqualifier;
2 import javax.enterprise.context.ApplicationScoped;
3
4 @Named(value = "multipleQualifier")
5 @ApplicationScoped
6 public class QualifierManager {
7
8     @Inject
9     @EQualifier(EDatabaseName.POSTGRESQL)
10    IDatabaseName databaseName;
11
12    public String getDatabaseName() {
13        return databaseName.databaseName();
14    }
15
16 }
17
```


— Stereotypes

Bir sınıf stereotipler ile ilişkilendirildiğinde Spring kütüphanesi otomatik olarak sınıfı context'e atar. Bunun anlamı ise; Spring kontrolü eline aldığı anda context'de ki beaniniz ile diğer beanler arasında dependency injection uygulayabilir.

MultipleAnotation.java X

```
1 package com.innova.databaseHomework.stereotypes;
2
3 import static java.lang.annotation.ElementType.FIELD;
4
15
16 @Stereotype
17 @Target({ TYPE, METHOD, FIELD })
18 @Retention(RUNTIME)
19 @Documented
20
21 @Named
22 @ApplicationScoped
23 public @interface MultipleAnotation {
24
25 }
26
```

DatabaseBean.java X

```
1 package com.innova.databaseHomework.stereotypes;
2
3 @MultipleAnotation
4 public class DatabaseBean {
5
6     private String name="Postgresql";
7
8     public String getName() {
9         return name;
10    }
11
12    public void setName(String name) {
13        this.name = name;
14    }
15
16
17
18 }
19
```

Interceptor

Interceptor'ler bize metot çağrıldığında, öncesinde ve sonrasında araya girerek belirli işlemleri yapabilmemizi sağlayan yapılardır.

!

InterceptorMethod.java

```

1 package com.innova.databaseHomework.interceptor;
2
3 import javax.interceptor.AroundInvoke;
4
5
6 @Interceptor
7 @InterceptorInterface
8 public class InterceptorMethod {
9
10
11     @AroundInvoke
12     public Object aroundInvoke(InvocationContext context) throws Exception
13
14         System.out.println("önceki durumum " + context.getMethod().getName()
15
16         // yol kesici hani nerde
17         // database sorgulama yaptık, session ?
18         boolean isLogin = false;
19         Object isContinue = null;
20         if (isLogin) {
21             System.out.println("Öncelikle üye olunuz !!! üye sayfasına yönlendirme yapılıyor");
22             return null;
23         } else {
24             try {
25                 isContinue = context.proceed(); // mühürledik artık devam edelim
26                 System.out.println("Login olduktan sonra : Yönlendirme yapılıyor");
27             } catch (Exception e) {
28                 e.printStackTrace();
29             }
30         }
31         return isContinue;
32     }
33 }

```

AopBean.java

```

8
9 @Named("interceptorAOP")
10 @ApplicationScoped
11 public class AopBean implements Serializable {
12
13     @Inject
14     private Login login;
15
16     public String getLogin() {
17         return login.isLogin();
18     }
19 }

```

Login.java

```

1 package com.innova.databaseHomework.interceptor;
2
3 @InterceptorInterface
4 public class Login {
5
6     public String isLogin() {
7         return "something";
8     }
9 }

```

InterceptorInterface.java

```

14
15 @InterceptorBinding
16 @Inherited
17 @Target({ TYPE, METHOD })
18 @Retention(RUNTIME)
19 @Documented
20
21 public @interface InterceptorInterface {
22
23 }

```

Projenin Github Linki:

<https://github.com/htcoztrk/InnovaSpringBootcamp/tree/master/com.innova.homework>