

Compte Rendu

Activité pratique N°1

**Inversion de contrôle et Injection
des dépendances**

EL GHALBZOURI HASSAN

Énoncé

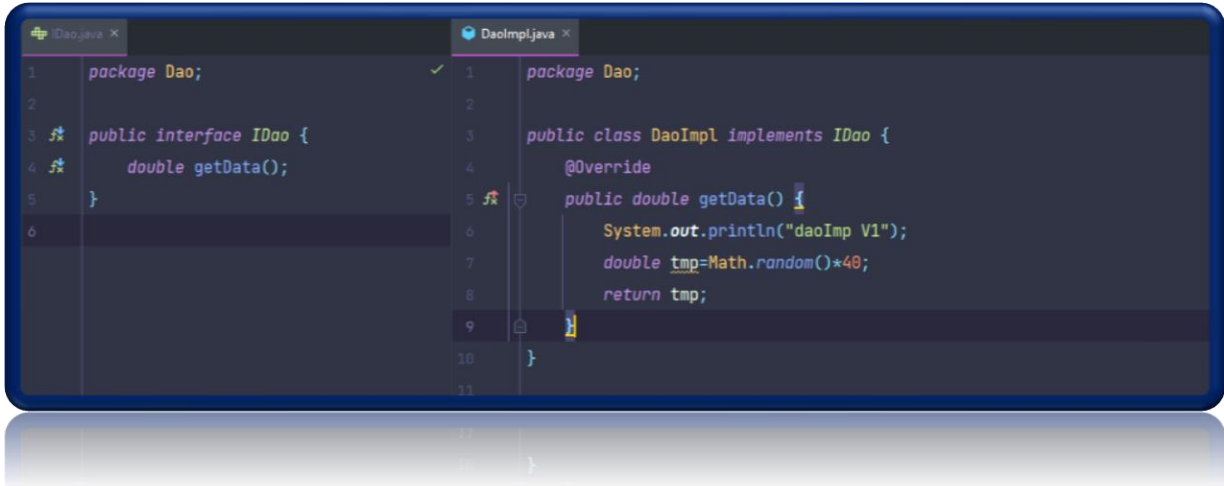
Inversion de contrôle et Injection des dépendances :

Rendre un compte rendu en reprenant l'exemple traité dans les vidéos des deux dernières séances

1. Créer l'interface IDao
2. Créer une implémentation de cette interface
3. Créer l'interface IMetier
4. Créer une implémentation de cette interface en utilisant le couplage faible
5. Faire l'injection des dépendances :
 - a. Par instanciation statique
 - b. Par instanciation dynamique
 - c. En utilisant le Framework Spring
 - Version XML
 - Version annotations

Code

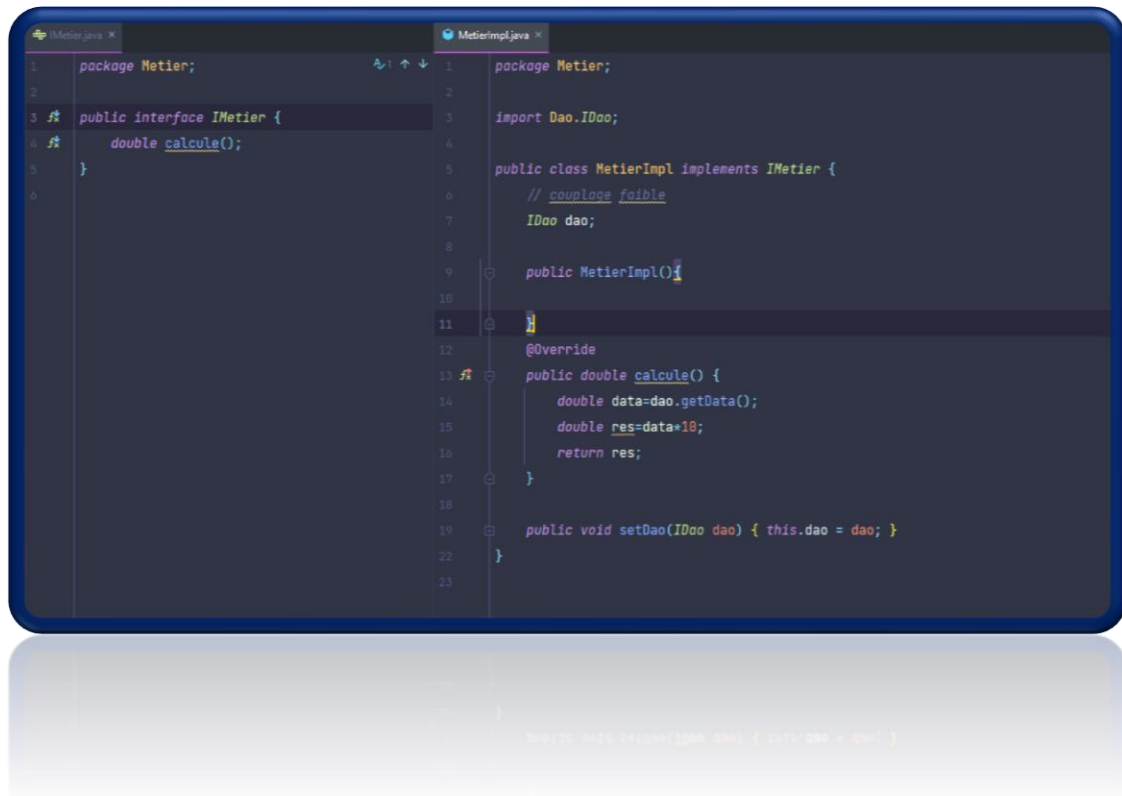
- L'interface IDao et Implémentation :
 - getData() : permet de retourner un donnée aléatoire



```
1 package Dao;
2
3 public interface IDao {
4     double getData();
5 }
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

```
1 package Dao;
2
3 public class DaoImpl implements IDao {
4     @Override
5     public double getData() {
6         System.out.println("daoImp V1");
7         double tmp=Math.random()*40;
8         return tmp;
9     }
10 }
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

- L'interface IMetier et l'implémentation par l'utilisation de couplage faible
 - Calcule () : permet de faire un petit calcul par l'utilisation de méthode getData()



```
1 package Metier;
2
3 public interface IMetier {
4     double calcule();
5 }
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

```
1 package Metier;
2
3 import Dao.IDao;
4
5 public class MetierImpl implements IMetier {
6     // couplage faible
7     IDao dao;
8
9     public MetierImpl() {
10
11     }
12
13     @Override
14     public double calcule() {
15         double data=dao.getData();
16         double res=data*10;
17         return res;
18     }
19
20     public void setDao(IDao dao) { this.dao = dao; }
21
22 }
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

- L'injection des dépendances :
 - Par instantiation statique :

The screenshot shows an IDE with a project named "1-Activité pratique N°1 [ActivitePratique1]". The package structure includes "InjectionDesDependances", "Dynamique", "SpringAnnotations", "SpringXML", "Statique", "Metier", and "resources". The "Statique.java" file is open, showing the following code:

```

1 package InjectionDesDependances;
2
3 import Dao.DaoImpl;
4 import Dao.IDao;
5 import Metier.MetierImpl;
6
7 public class Statique {
8     public static void main(String[] args) {
9         IDao dao=new DaoImpl();
10        MetierImpl metier=new MetierImpl();
11        metier.setDao(dao);
12        System.out.println("----->Statique");
13        System.out.println(metier.calculer());
14    }
15 }

```

The console output shows the execution of the program, with the message "Process finished with exit code 0".

- Par instantiation dynamique :

The screenshot shows the same IDE with the "Dynamique.java" file open. The code is as follows:

```

1 public class Dynamique {
2     public static void main(String[] args) throws Exception {
3
4         Scanner scanner=new Scanner(new File("config.txt"));
5         String daoClassName=scanner.nextLine();
6         Class cDao=Class.forName(daoClassName);
7         IDao dao=(IDao) cDao.newInstance();
8
9         String metierClassName=scanner.nextLine();
10        Class cMetier=Class.forName(metierClassName);
11        IMetier metier=(IMetier) cMetier.newInstance();
12
13        Method methodSetDao=cMetier.getMethod("setDao",IDao.class);
14        methodSetDao.invoke(metier,dao);
15
16        System.out.println("----->Dynamique");
17        System.out.println(metier.calculer());
18    }
19 }

```

The console output shows the execution of the program, with the message "Process finished with exit code 0".

- L'injection des dépendances en utilisant le Framework Spring :
 - Version XML :

The screenshot shows an IDE with two main windows. The left window displays a Java class named `SpringXML` in the package `InjectionDesDependances`. It imports `Metier`, `ApplicationContext`, and `ClassPathXmlApplicationContext`. The `main` method creates an `ApplicationContext` with `context.xml` as the configuration location, retrieves a `Metier` bean, and prints its `calculer()` result. The right window shows the `context.xml` file, which is an XML configuration for Spring. It defines two beans: `dao` of type `Dao.DaoImpl` and `metier` of type `Metier.MetierImpl`, with the `metier` bean depending on the `dao` bean.

- Version annotations :
 - Dans ce cas j'ai ajouté deux class `MetierImplV2` et `DaoImplV2` implémentés respectivement de `IMetier` et `IDao` pour utiliser les annotations

The screenshot shows an IDE with two Java files. The left file is `MetierImplV2`, which implements the `IMetier` interface. It uses `@Component("metier")` and `@Autowired` to inject an `IDao` bean. The `calculer()` method uses the injected `dao` to get data and calculate a result. The right file is `DaoImplV2`, which implements the `IDao` interface. It uses `@Component("dao")` and `@Override` to implement the `getData()` method, which returns a random value between 0 and 10.

