



**ECOLE MAROCAINE DES
SCIENCES DE L'INGENIEUR**

Membre de

HONORIS UNITED UNIVERSITIES

Rapport De Travaux Pratiques

Encadré par :

- Pr.BADRI Tijane

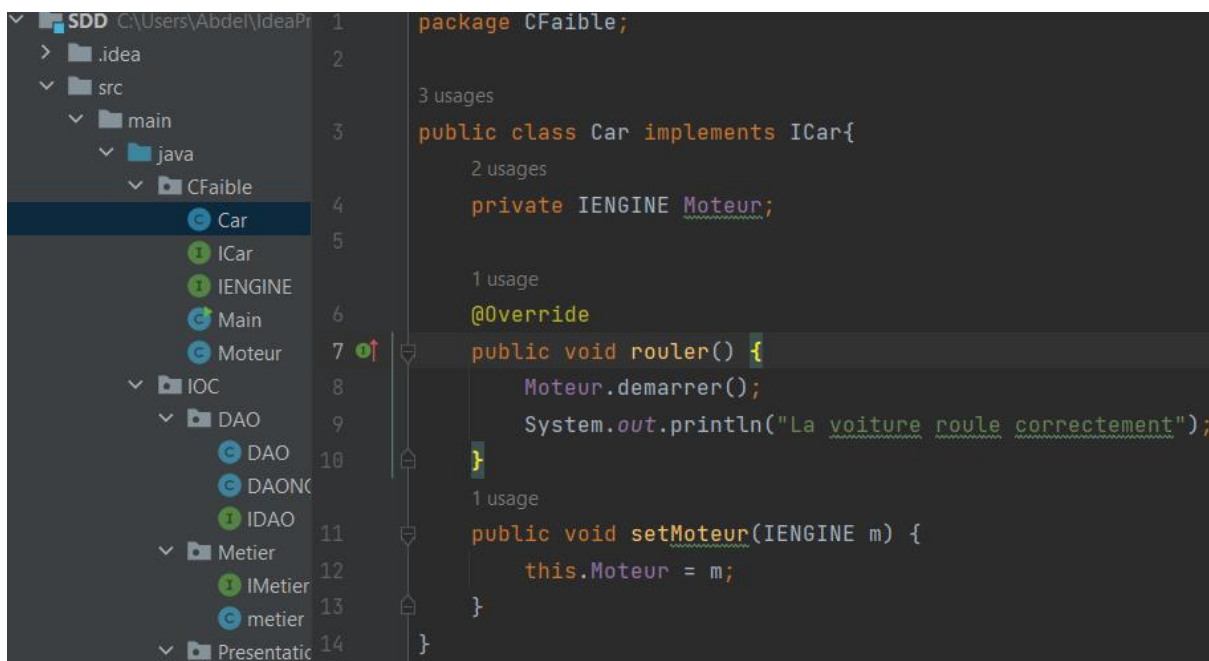
Réalisé par :

- TAWFIK Abdelmoughit

Couplage Faible

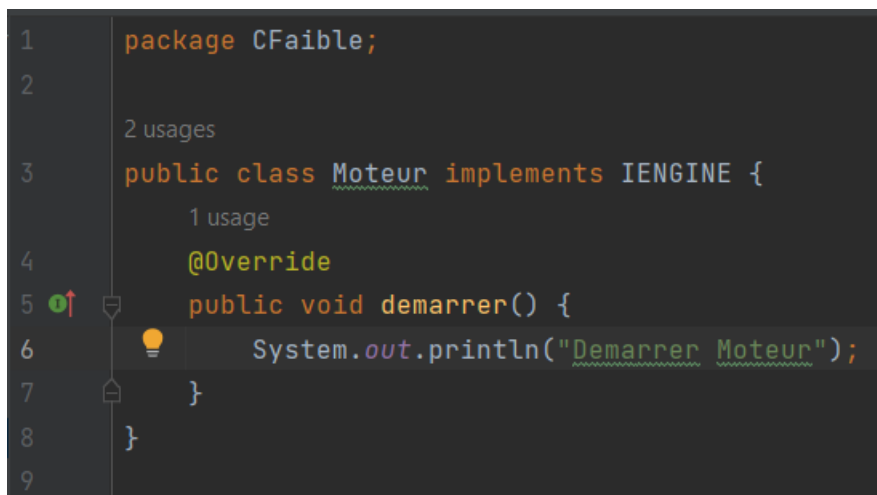
L'idée générale du couplage faible consiste à établir un protocole d'échange et à effectuer le moins d'hypothèses (ou à imposer le moins de contraintes) possible entre les composants . Ainsi, les composants interagissent dans un cadre défini.

Exemple :



```
1 package CFaible;
2
3 3 usages
4 public class Car implements ICar{
5     2 usages
6     private IENGINE Moteur;
7
8     1 usage
9     @Override
10    public void rouler() {
11        Moteur.demarrer();
12        System.out.println("La voiture roule correctement");
13    }
14
15    1 usage
16    public void setMoteur(ENGINE m) {
17        this.Moteur = m;
18    }
19 }
```

The screenshot shows an IDE with a project structure on the left and a code editor on the right. The project structure includes a package CFaible with classes Car, ICar, IENGINE, Main, Moteur, and IOC. The code editor displays the Car class implementation, which implements the ICar interface. It has a private IENGINE Moteur attribute and two methods: rouler() and setMoteur(). The rouler() method calls Moteur.demarrer() and prints a message. The setMoteur() method sets the Moteur attribute to the passed parameter m.



```
1 package CFaible;
2
3 2 usages
4 public class Moteur implements IENGINE {
5     1 usage
6     @Override
7     public void demarrer() {
8         System.out.println("Demarrer Moteur");
9     }
10 }
```

The screenshot shows the Moteur class implementation in the CFaible package. It implements the IENGINE interface and has a single method demarrer() which prints "Demarrer Moteur".

```

1  package CFaible;
2
3  import CFaible.Car;
4  import CFaible.Moteur;
5
6  no usages
7  ▶ public class Main {
8  ▶  no usages
9  ▶  public static void main(String[] args) {
10 ▶      Car v = new Car();
11      v.setMoteur(new Moteur());
12      v.rouler();
13      System.out.println("Bon Voyage !");
14  }
15  }

```

```

1  package CFaible;
2
3  3 usages  1 implementation
4  1 usage  1 implementation
5  public interface IENGINE {
6      void demarrer();
7  }

```

Le **couplage** est une métrique indiquant le niveau d'interaction entre deux ou plusieurs composants logiciels. Deux composants sont dits couplés s'ils échangent de l'information.

On parle de **couplage fort** ou **couplage serré** si les composants échangent beaucoup d'information. On parle de **couplage faible**, **couplage léger** ou **couplage lâche** si les composants échangent peu d'information et/ou de manière désynchronisée.

Injection des dépendances

1. Instanciation Statique

```

1 package IOC.DAO;
2
3 public class DAO implements IDAO{
4     @Override
5     public double getData() {
6         System.out.println("From SQL DB");
7         return (7);
8     }
9 }
10

```

```

1 package IOC.DAO;
2
3 public interface IDAO {
4     double getData();
5 }

```

```

1 package IOC.DAO;
2
3 public class DAONOSQL implements IDAO {
4     @Override
5     public double getData() {
6         System.out.println("From No SQL DB");
7         return (10);
8     }
9 }

```

```
1 package IOC.Metier;
2
3 import IOC.DAO.IDAO;
4
5 3 usages
6 public class metier implements IMetier {
7     2 usages
8     IDAO dao;
9     2 usages
10    @Override
11    public double calcul() {
12        double data = dao.getData();
13        return data*10;
14    }
15    1 usage
16    public void setDao(IDAO dao) {
17        this.dao = dao;
18    }
19 }
```

2.Instanciacion Dynamique

```
1 package IOC.Presentation;
2
3 import IOC.DAO.IDAO;
4 import IOC.Metier.IMetier;
5
6 import java.io.File;
7 import java.lang.reflect.Method;
8 import java.util.Scanner;
9
10 no usages
11 public class Dynamique {
12     no usages
13     public static void main(String[] args) throws Exception {
14         Scanner sc = new Scanner(new File("src/IOC/Config.txt"));
15         String dao = sc.nextLine();
16         Class clsDao = Class.forName(dao);
17         IDAO objDao = (IDAO) clsDao.newInstance();
18
19         String metier = sc.nextLine();
20         Class clsMetier = Class.forName(metier);
21         IMetier objMetier = (IMetier) clsMetier.newInstance();
22
23         Method method = clsMetier.getMethod("setDao", IDAO.class);
24         method.invoke(objMetier, objDao);
25
26         System.out.println(objMetier.calcul());
27     }
28 }
```