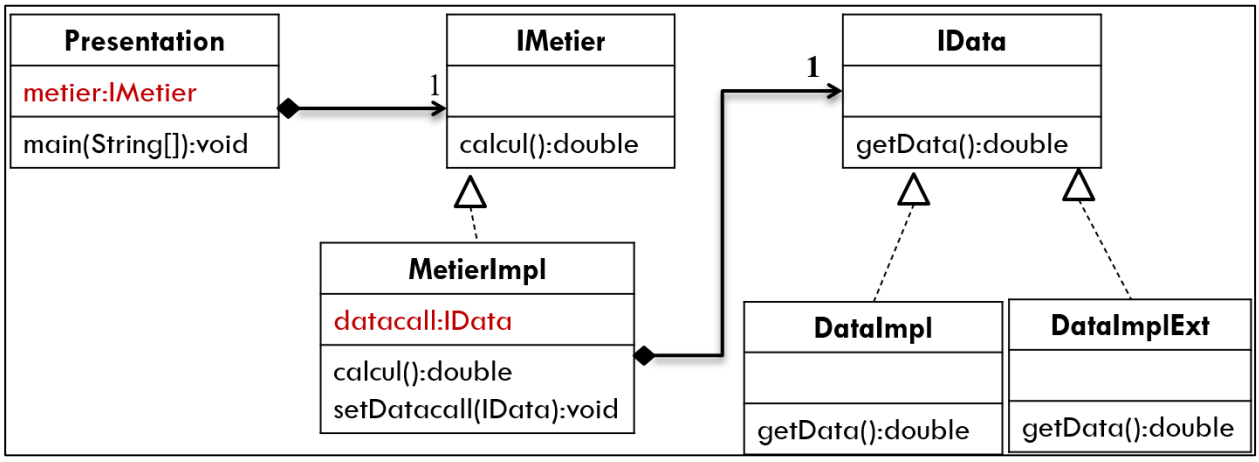
TP 2 - Atelier Couplage Fort Vs Couplage Faible (IOC et ID) :

Partie I : Fichier de configuration

1. Reproduire le même projet abordé dans la séance de couplage fort et couplage faible (instanciation statique et instanciation dynamique), en respectant l'architecture suivante :



2. Donner des limitations concernant l'utilisation d'un fichier « .txt ».
3. Donner un exemple.

On propose d'utiliser un fichier « .xml » afin de remédier à ces limitations. Spring implémente cette solution en proposant une structure bien définie.

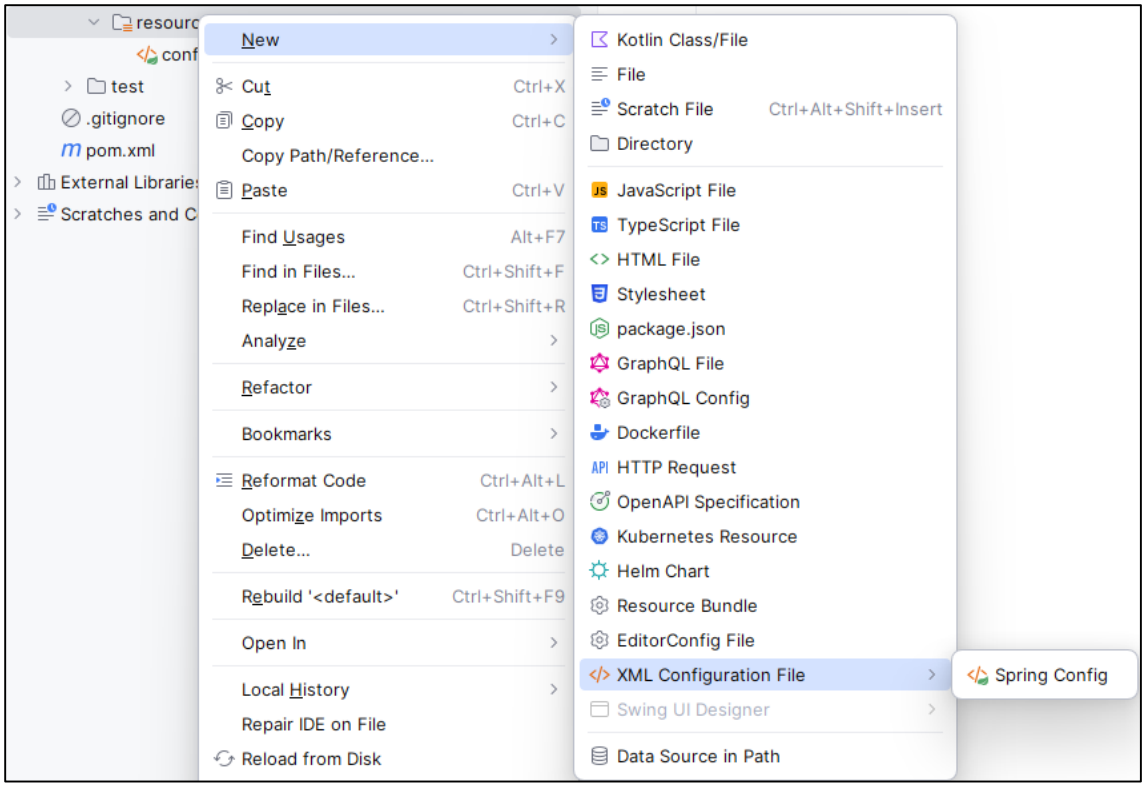
Avant de procéder, il est nécessaire de rajouter les dépendances de base de spring sur votre projet.

4. Dans le fichier **pom.xml**, ajouter les dépendances suivantes :
   4.1. Spring-core
   4.2. Spring-Context
   4.3. Spring-beans

Une fois le projet configuré et les dépendances de spring installées, on procède l'implémentation de la nouvelle solution avec un couplage faible.

Spring propose de définir un fichier de configuration **«.xml»**

5. Dans le dossier **resources**, rajouter un fichier de type **Spring Config**:



6. Sur ce fichier crée, Rajouter les lignes [3-6] suivantes :

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">
3. <bean id="data" class="data.DataImpl"></bean>
4. <bean id="metier" class="metier.MetierImpl">
5.       <property name="datacall" ref="data"></property>
6.  </bean>
    </beans>
```

7. Dans le package presentation créer la classe PresSpringXml qui contient la méthode main:

```
public class PresSpringXml {
   public static void main(String[] args) {
      ApplicationContext applicationContext = new ClassPathXmlApplicationContext("config.xml");
      IMetier metier = (IMetier) applicationContext.getBean("metier");
      System.out.println(metier.calcule());
   }
}
```

8. On suppose que la classe « metierImpl » gère l'injection des dépendances via un constructeur à un seul argument.

   Question : Expliquer les changements à apporter au niveau de la solution spring via le fichier « xml »

Partie II : Les annotations avec Spring

Dans le cadre du principe IOC, permettant de « Déléguer la mise en place des exigences techniques au Framework » ce TP prendra en charge le « Framework spring ».

Spring propose une multitude d'annotations pour faciliter le travail pour les développeurs afin de se concentrer sur la partie fonctionnelle tout en assurant un couplage faible.

1. Au niveau des classes des objets qu'on va utiliser par la suite, on rajoute directement l'annotation « @Component »

```
@Component
public class DataImpl implements IData {
    @Override
    public double getData() {
        System.out.println("Recuperation de la Base de donnes ");
        double data = 10;
        return data;
    }
}
```

2. On rajoute la même annotation au niveau des implémentations de l'interface « IMetier »
3. Afin d'assurer l'injection des dépendances » mise en œuvre au niveau des solutions précédentes. Spring propose de rajouter l'annotation « @Autowired »
   a. Rajouter cette annotation comme suit :

```
@Component
public class MetierImpl implements IMetier {
    @Autowired
    private IData datacall;
    @Override
    public double calcule() {
        double data = datacall.getData();
        double result = data * 15.6;
        return result;
    }
    public void setDatacall(DataImpl datacall) {
        this.datacall = datacall;
    }
}
```

4. Finalement, rajouter une nouvelle classe PresSpringAnnotation.

a. La mise en œuvre des changements apportés se présente ainsi :

```
public class AnnotationConfigPresentation {
    public static void main(String[] args) {
        ApplicationContext applicationContext = new AnnotationConfigApplicationContext("metier" , "data");
        IMetier metier = (IMetier) applicationContext.getBean(IMetier.class);
        System.out.println(metier.calcule());
    }
}
```

Exécuter la fonction main dans la classe. Les deux informations, passées en paramètre au constructeur de la classe **AnnotationConfigApplicationContext,** représentent les deux packages qui englobent, respectivement, les classes des objets annotés par **@Component**.

5. On suppose que le même package **data** contient deux implémentations de l'interface IData. Le nom du Package n'est plus suffisant pour définir la classe à utiliser, et ce comme précisé sur la couche présentation.
    a. Afin de lever la confusion, l'annotation @Component sera accompagnée par une étiquette.

```
@Component("data")
public class DataImpl implements IData {
    @Override
    public double getData() {
        ...
```

Appliquer la même méthode sur les classes présentant une confusion.

6. Rajouter l'annotation **@Qualifier** au niveau de la classe **MetierImpl,** pour définir la classe choisie :

```
@Component
public class MetierImpl implements IMetier {
    @Autowired
    @Qualifier("data")
    private IData datacall;
...
}
```

7. Tester l'application.