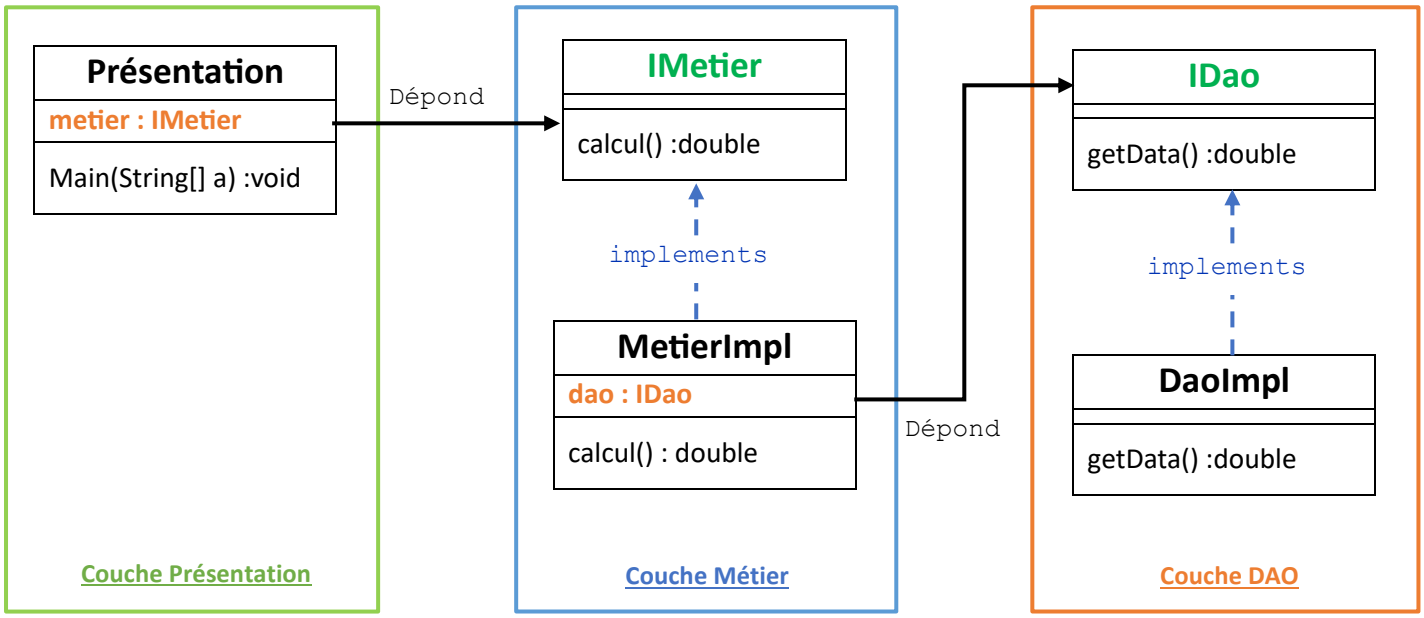
Mise en œuvre de l'Inversion de Contrôle et Injection de dépendances

Notions de base :

- **Classe Abstraite** est une classe qui peut contenir les attributs, les méthodes et les méthodes abstraites(méthodes obligées de redéfinir dans les classes dérivé).
- **Interface** est une classe abstraite qui ne contient que les méthodes abstraites
  - Toutes les méthodes d'une interface par défaut sont publiques
- **Extension** est une classe qui implémente une interface qui existe déjà dans le logiciel
- **new = Couplage fort (fait appel à une classe)**
- Dans java un objet n'est pas initialisé sa valeur par défaut est **null**

Application :



- **Injection de dépendances** par :
  - Instanciation statique ➔ **new**

```
public class Presentation {
    public static void main(String[] args){
        //Injection des dépondances par instanciation statique ==> new
        DaoImpl2 dao = new DaoImpl2();
        MetierImpl metier = new MetierImpl();
        metier.setDao(dao);
        System.out.println("Résultat = "+metier.clacul());
    }
}
```

- Instanciation dynamique

```
String daoClasseName = scanner.nextLine();
//Demande de charger une classe dans la mémoire
Class cDao = Class.forName(daoClasseName);
IDao dao = (IDao) cDao.newInstance();

String metierClasseName = scanner.nextLine();
Class cMetier = Class.forName(metierClasseName);
IMetier metier = (IMetier) cMetier.newInstance();

// comme metier.setDao(dao);
Method method = cMetier.getMethod( name: "setDao", IDao.class);
method.invoke(metier, dao);

System.out.println("Résulat = "+metier.clacul() );
```

config.txt

```
1  ext.DaoImplVWS
2  metier.MetierImpl
```

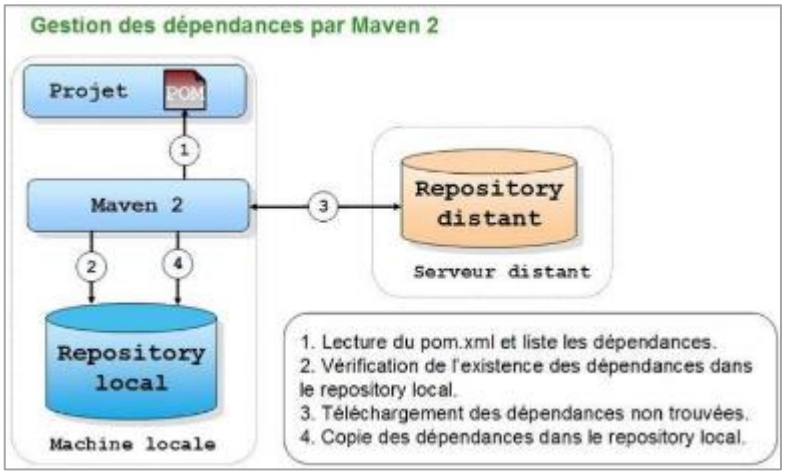
MetierImpl metier = new MetierImpl();

metier.setDao(dao) ;

- Framework Spring : pour travailler avec Spring nous somme besoin d'utiliser les librairies de Spring (il faut installer les jars), manuellement c'est difficile d'ajouter les jars au projet(on va être perdu entre ces jars) ➔ Alors utiliser un outil qui va gérer les dépendance ➔ Maven ou Gradle(Android studio)

Maven: POM (Project Object Model)

- Maven est un outil de construction de projets (**build**)
- Un outil permettant d'automatiser la gestion de projets Java :
  - Compilation et déploiement des applications Java
  - Gestion des librairies requises par l'application
  - Exécution des tests unitaires
  - Génération des documentations du projet…



- **mvn compile ➔ compiler le code**
- **mvn test➔ compiler + exécuter les tests unitaire**
- **mvn package ➔ compiler + exécuter les tests + générer un fichier .jar**(java archive) **ou .war**(web archive)
- **mvn install ➔ … installer le .jar dans le repo local(.m2)**
- **mvn deploy ➔ déployer le .war dans un serveur d'application**
- **mvn site ➔ générer des fichiers html dont on trouve la documentation de projet(java doc)**

Maven Project:

> **SNAPSHOT :** Signifie en cours de test

> **Version XML :**

```
<bean id="dao" class="kb.practice.dao.DaoImpl"></bean>
<bean id="metier" class="kb.practice.metier.MetierImpl">
    <property name="dao" ref="dao"></property>
</bean>
```

setDao(dao)

```
<bean id="dao" class="kb.practice.dao.DaoImpl"></bean>
<bean id="metier" class="kb.practice.metier.MetierImpl">
    <constructor-arg ref="dao"></constructor-arg>
</bean>
```

Constricteur avec paramètre dao

```
public class PresSpringXML {
    public static void main(String[] args) {
        ApplicationContext context =
                new ClassPathXmlApplicationContext( configLocation: "applicationContext.xml");
        IMetier metier = (IMetier) context.getBean( s: "metier");
        System.out.println("Résultat = "+metier.clacul());
    }
}
```

```
Version normale BD
Résultat = -23872.465422500056
```

> **Version annotation**

- Spring au démarrage à chaque fois que tu trouve par l'annotation component il va l'instancier et il va le donner comme nom dao

```
@Component("dao3")
public class DoaImplVWS implements IDao {
```

Note :

Spring au moment qu'il va instancier la classe MetierImpl il cherche s'il trouve un objet de type IDao il va l'injecter dans le variable dao

- **Injection via l'annotation Autowired**

```
@Component
public class MetierImpl implements IMetier {
    //Injection des dépendances via
    @Autowired
    @Qualifier("dao3")
    private IDao dao; // Couplage faible
```

- **Injection via le constructeur**

```
//Injection des dépendances via le constructeur
public MetierImpl(IDao dao) {
    this.dao = dao;
}
```

⇨ **La première instance d'une implémentation de l'interface IDao crée dans la mémoire va être instancié à l'objet dao**

```
public class PresSpringAnnotations {
    public static void main(String[] args) {
        // va scanner tous les sous-packages de package kb
        ApplicationContext context = new AnnotationConfigApplicationContext( ...basePackages: "kb");
        IMetier metier = context.getBean(IMetier.class);
        System.out.println(metier.clacul());
    }
}
```