



# Cours Spring

## 02 – 03 Injection de dépendances en XML

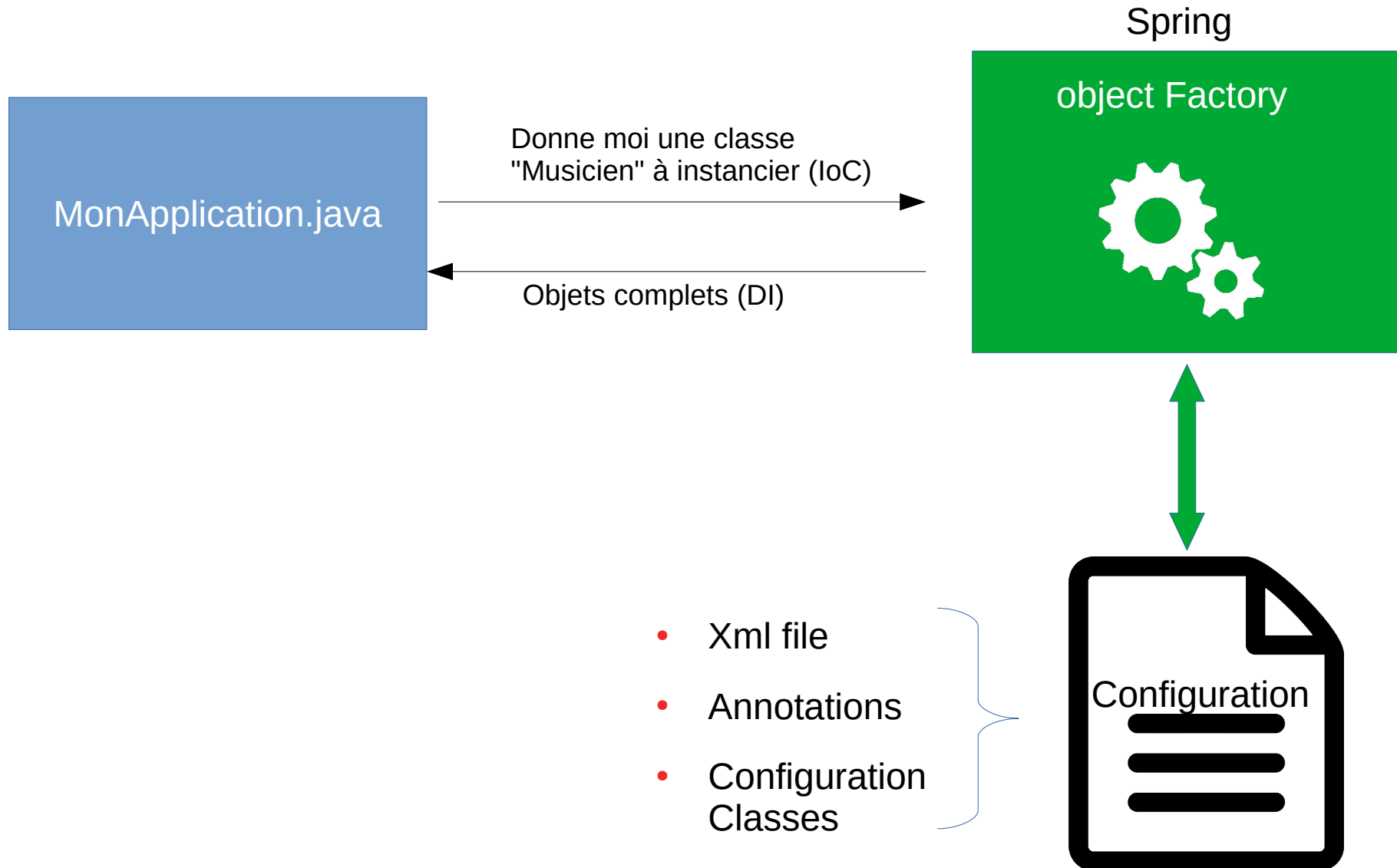
# Définition : Dependency injection (DI)

- Une dépendance d'un objet envers un autre signifie qu'un objet nécessite dans sa composition la présence d'un autre

Ex : un objet Voiture nécessite la présence d'un objet Moteur dans sa composition

- Spring doit créer des objets complets (en y incluant d'éventuels objets imbriqués), prêts à l'emploi.

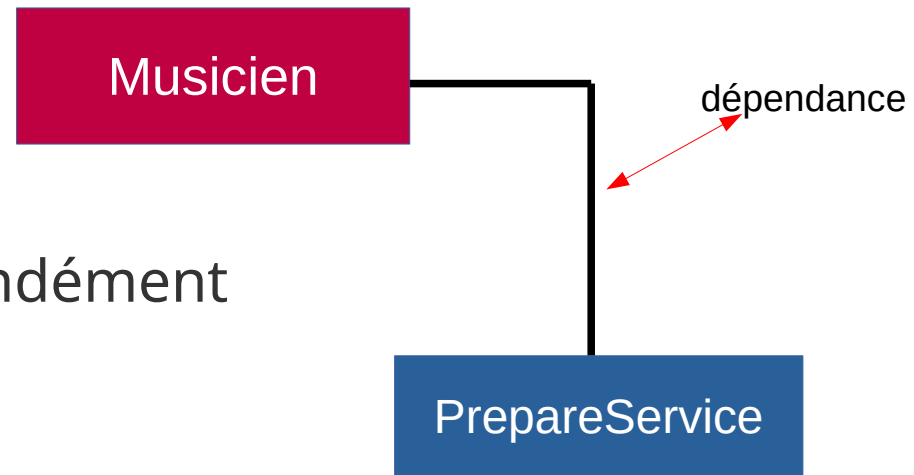
# Spring core : 2 fonctionnalités principales



# Poursuivons notre exemple

- Considérons qu'un musicien doit pouvoir se préparer avant une potentielle interprétation :

- Je m'installe tranquillement
- Je fais des gammes
- Je me relaxe, en respirant profondément
- Je me concentre
- ...etc



- En utilisant un service : prepareService



# 3 types d'injection de dépendance

- Par constructeur
- Par setter()
- Par "autowiring"

On regarde d'abord les deux premières en xml

- "L'autowiring" sera abordé via les Annotations

# L'injection par constructeur (début)

- Préalable : définir le service (classe + interface)

```
public class ZenPreparationService implements PrepareService {  
    @Override  
    public String getPreparation() {  
        return "Je me relaxe en respirant profondément";  
    }  
}
```

```
public interface PrepareService {  
    public String getPreparation();  
}
```

- Ajouter la fonctionnalité aux Musiciens (impl + interfaces)

```
public class Violoniste implements Musicien {  
    @Override  
    public String joueTaPartition(){  
        return "je joue de la Valse d'Amélie";  
    }  
  
    @Override  
    public String getPrepa() {  
        // TODO Auto-generated method stub  
        return null;  
    }  
}
```

```
public class Pianiste implements Musicien{  
    @Override  
    public String joueTaPartition() {  
        return "je joue du piano debout";  
    }  
  
    @Override  
    public String getPrepa() {  
        // TODO Auto-generated method stub  
        return null;  
    }  
}
```

```
public interface Musicien {  
    public String joueTaPartition();  
    public String getPrepa();  
}
```

# Di par Constructeur (suite)

- Dans la classe qui doit posséder la dépendance, créer un constructeur, et un champ pour y stocker la dépendance
- On utilise le service dans la méthode getPrepa()

```
public class Violoniste implements Musicien {
    @Override
    public String joueTaPartition(){
        return "je joue de la Valse d'Amélie";
    }

    private PrepareService service ;

    public Violoniste(PrepareService unService) {
        service = unService;
    }

    @Override
    public String getPrepa() {
        return service.getPreparation();
    }
}
```

```
public class Pianiste implements Musicien{
    @Override
    public String joueTaPartition() {
        return "je joue du piano debout";
    }

    private PrepareService service ;

    public Pianiste(PrepareService unService) {
        service = unService;
    }

    @Override
    public String getPrepa() {
        return service.getPreparation();
    }
}
```

# DI par constructeur (fin)

- Configurer la dépendance dans le fichier xml de configuration du contexte : [applicationContext.xml](#)

Déclaration de la dépendance qui fournit la fonctionnalité

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans . . . ">

<bean id="unPrepareService"
      class="com.springdemo.ZenPreparationService" >
</bean>

<bean id="unMusicien" class="com.springdemo.Violoniste" >
  <constructor-arg ref="unPrepareService"/>
</bean>

</beans>
```

Injection par constructeur de la dépendance

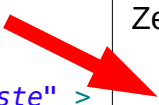


# En arrière plan

- Spring réalise des opérations pour vous en arrière plan , en se basant sur le fichier de configuration.

```
<bean id="unPrepareService"  
      class="com.springdemo.ZenPreparationService" >  
</bean>
```

```
<bean id="unMusicien" class="com.springdemo.Violoniste" >  
  <constructor-arg ref="unPrepareService"/>  
</bean>
```



```
ZenPreparationService preparationService =  
    new ZenPreparationService() ;
```

```
Violoniste musicien = new Violoniste( preparationService) ;
```

# On teste notre injection

- Dans la methode main() on appelle la méthode du service injecté via le musicien

```
public class MonApplication {  
    public static void main(String[] args) {  
        //load the spring configuration file  
        ClassPathXmlApplicationContext context = new  
        ClassPathXmlApplicationContext("applicationContext.xml");  
  
        //retrieve a beam from the spring container  
        Musicien musicien = context.getBean("monMusicien",  
        Musicien.class);  
  
        //use the objet  
        System.out.println( musicien.joueTaPartition());  
  
        //use the DI  
        System.out.println( musicien.getPrepa());  
  
        // close the application context  
        context.close();  
    }  
}
```

# Conclusion DI avec xml

- Si on execute l'aplication =>

A screenshot of an IDE's console window. The window has a title bar with tabs for 'Problems', 'Javadoc', 'Declaration', and 'Console'. The 'Console' tab is active. The output text in the console is: '<terminated> MonApplication (4) [Java Application] /Library/Java/JavaVirtualMachines/oper', 'je joue de la Valse d'Amélie', and 'Je me relaxe en respirant profondément'.

```
<terminated> MonApplication (4) [Java Application] /Library/Java/JavaVirtualMachines/oper
je joue de la Valse d'Amélie
Je me relaxe en respirant profondément
```

- On a bien configuré IOC et Di via la méthode xml.