

# Le framework Spring

## Module 3 – Spring Core



# Objectifs

- Couplage faible vs couplage fort
- Injection de dépendance et inversion de contrôle
- Notion de Spring bean
  - Définir un bean avec @Component, @Controller, @Service
  - Définir un bean par programmation avec @Configuration et @Bean
- Différents types d'injection :
  - Par setter / Par constructeur / par attribut
  - Par type ou par nom
- Gérer un conflit
  - @Qualifier / @Primary / @Profile

# Problématique du couplage

- Couplage fort / couplage faible
  - 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
  - Le **couplage fort** implique une dépendance forte entre deux composants
    - Difficilement réutilisable
    - Difficilement testable
  - Le **couplage faible** favorise :
    - La faible dépendance entre les classes
    - La réduction de l'impact des changements dans une classe
    - La réutilisation des classes ou modules

Spring Core

# Couplage fort / Couplage faible

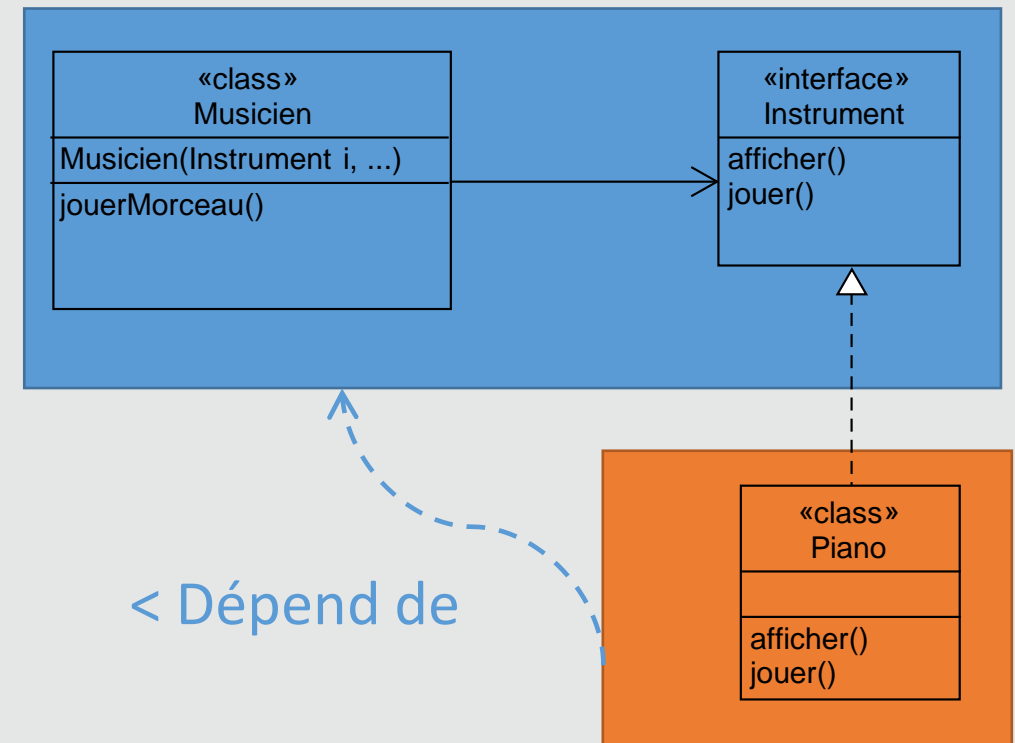
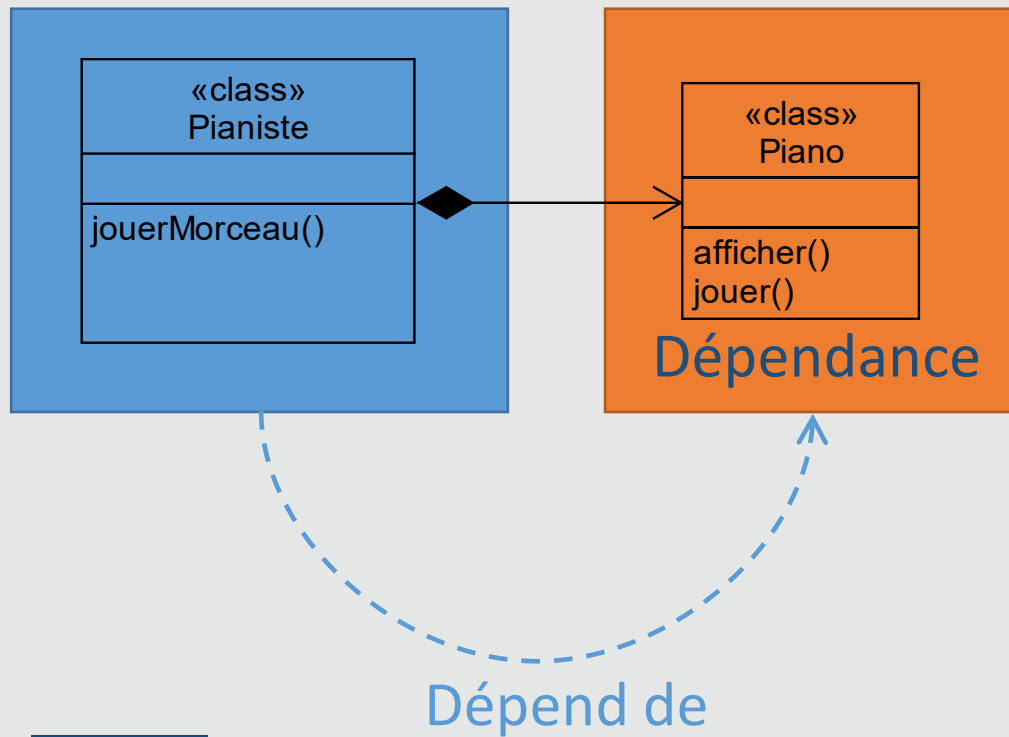
## Démonstration



# L'inversion de contrôle

- Inversion of Control (IoC)
  - Réduire les dépendances (couplage) entre des objets dont l'implémentation peut varier
  - Diminuer la complexité de gestion du cycle de vie de ces objets (patterns Singleton et Factory)
  - Le contrôle du flot d'exécution d'une application n'est plus géré par l'application elle-même mais par une structure externe (conteneur)
- Mise en place
  - Utilisation du patron de conception (design pattern) Factory
  - Utilisation de l'injection de dépendances.

# Inversion Of Control : Exemple




# L'injection de dépendances

- Dependency Injection (DI)
- Mécanisme permettant d'implémenter le principe de l'inversion de contrôle
  - Permet d'éviter une dépendance directe entre deux classes en définissant dynamiquement la dépendance plutôt que statiquement
  - Permet à une application de déléguer la gestion du cycle de vie de ses dépendances et leurs injections à une autre entité
  - L'application ne crée pas directement les instances des objets dont elle a besoin : les dépendances d'un objet ne sont pas gérées par l'objet lui-même mais sont gérées et injectées par une entité externe à l'objet

# L'injection de dépendance

```
public class Musicien {  
    private String morceau;  
    private Instrument instrument;  
  
    public Musicien(String morceau, Instrument instrument) {  
        this.morceau = morceau;  
        this.instrument = instrument;  
    }  
}
```



```
public static void main(String[] args) {  
    System.out.println("Le pianiste : ");  
    Musicien pianiste = new Musicien("La 9eme de Beethoven", new Piano());  
    pianiste.jouerMorceau();  
  
    System.out.println("Le violoniste : ");  
    Musicien violoniste = new Musicien("La 9eme de Beethoven", new Violon());  
    violoniste.jouerMorceau();  
}
```



# Ce qu'apporte Spring

- Spring apporte le "conteneur léger"
  - Permet la prise en charge du cycle de vie des objets (beans) et leur mise en relation
  - Nommé : ApplicationContext
- Le noyau de Spring est basé sur :
  - Une fabrique générique de composants informatiques, composants nommés beans
  - Un conteneur capable de stocker ces beans

# Définir un bean par annotation

- Les beans définis par annotation doivent être placés au même niveau ou sous la classe d'application (classe annotée avec @SpringBootApplication)
- Les beans seront découverts lors d'un scan automatique des packages permis par l'annotation @SpringBootApplication
  - @SpringBootApplication ⇔
  - @Configuration
  - @EnableAutoConfiguration
  - @ComponentScan

# Définir un bean par annotation

- **@Component**
  - Annotation de base
- **@Controller**
  - Annotation sémantique désignant un composant de type contrôleur (MVC)
- **@Service**
  - Annotation sur un service métier – permet la gestion des transactions sur cette couche
- **@Repository**
  - Annotation sur une classe DAO – permet la gestion des exceptions et des transactions de cette couche

# Scope des beans

- 5 scopes
  - **singleton**
    - Scope par défaut
  - **prototype**
    - Une nouvelle instance à chaque injection
  - **request** (uniquement en environnement web)
    - Une nouvelle instance pour chaque requête HTTP
  - **session** (uniquement en environnement web)
    - Une nouvelle instance pour chaque nouvelle session
  - **application** (uniquement en environnement web)
    - Instance liée à un servlet context (ie une application web)

# Notions de Conteneur Spring et de bean

## Démonstration

# L'injection de dépendance par annotation

- @Autowired
- Cette annotation permet au conteneur Spring de rechercher un bean et de l'injecter
  - Dans une propriété,
  - Dans une méthode
  - Dans le constructeur

# Injection de dépendance avec Spring

## Démonstration



# Cas de conflit lors d'une injection par type

- Le bean `trainerController` demande l'injection d'un bean de type `TrainerService` par constructeur.
- 2 beans correspondent au type demandé : `TrainerServiceImpl` et `TrainerServiceMock`.
- Spring ne peut pas choisir, l'exécution aboutit à une erreur.



# Résoudre le conflit

Plusieurs solutions :

- Rendre un bean prioritaire avec l'annotation @Primary
- Transformer l'injection en une injection par nom avec @Qualifier
- Définir des profils ( @Profile ) et activer un des profils dans application.properties

# Conflit d'injection par type et Résolution

## Démonstration

Spring Core

# Escape Rooms

TP



# Définition des beans par programmation

- @Configuration et @Bean
- Centralise la définition des beans dans une même classe de configuration.
  - Centralise les dépendances
- Capacité à ajouter du code; pour des traitements supplémentaires :
  - Métiers,
  - Sécurité,
  - Gestion de plusieurs sources de données,
  - Utilisation de librairies externes,
  - ...

```
@Configuration
public class AppConfiguration {
    @Bean
    public TrainerDAO getBeanTrainerDAO() {
        return new TrainerDAOMock();
    }

    @Bean
    @Profile("default")
    public TrainerService getBeanTrainerService() {
        return new TrainerServiceImpl(getBeanTrainerDAO());
    }...
}
```

# Définition des beans par programmation

## Démonstration

Spring Core

# Filmothèque – Partie 01

TP



# Conclusion

- Spring Core préconise le couplage faible
- Spring Core s'appuie sur les design patterns :
  - Inversion de contrôle
  - Injection de dépendance
- Plusieurs types de beans
  - @Component, @Controller, @Service
- Différents types d'injection :
  - Par setter / Par constructeur / par attribut
  - Par type ou par nom
- Gestion de conflit
  - @Qualifier / @Primary / @Profile
- Configuration et définitions des beans par programmation pour ajouter du comportement