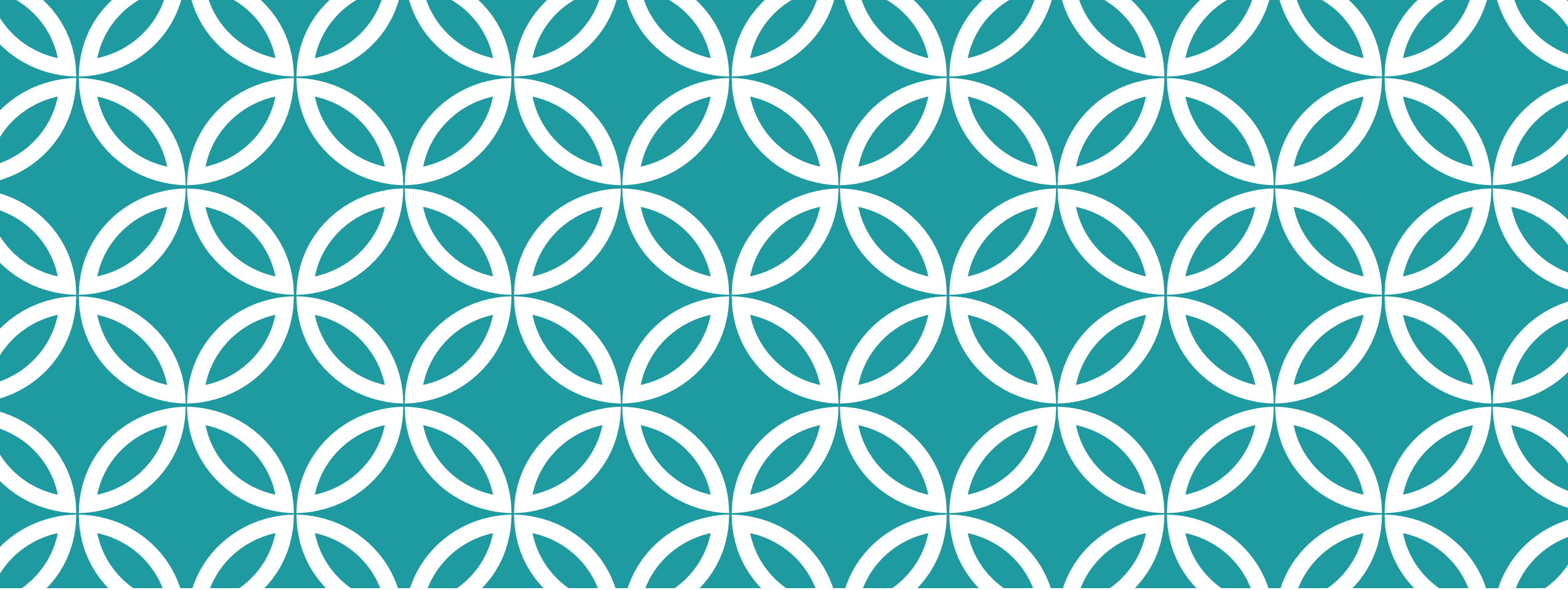
SPRING



SPRING CORE

Introduction à SPRING

PRÉSENTATION DE SPRING

Framework, standard industriel

Facilite le développement et les tests

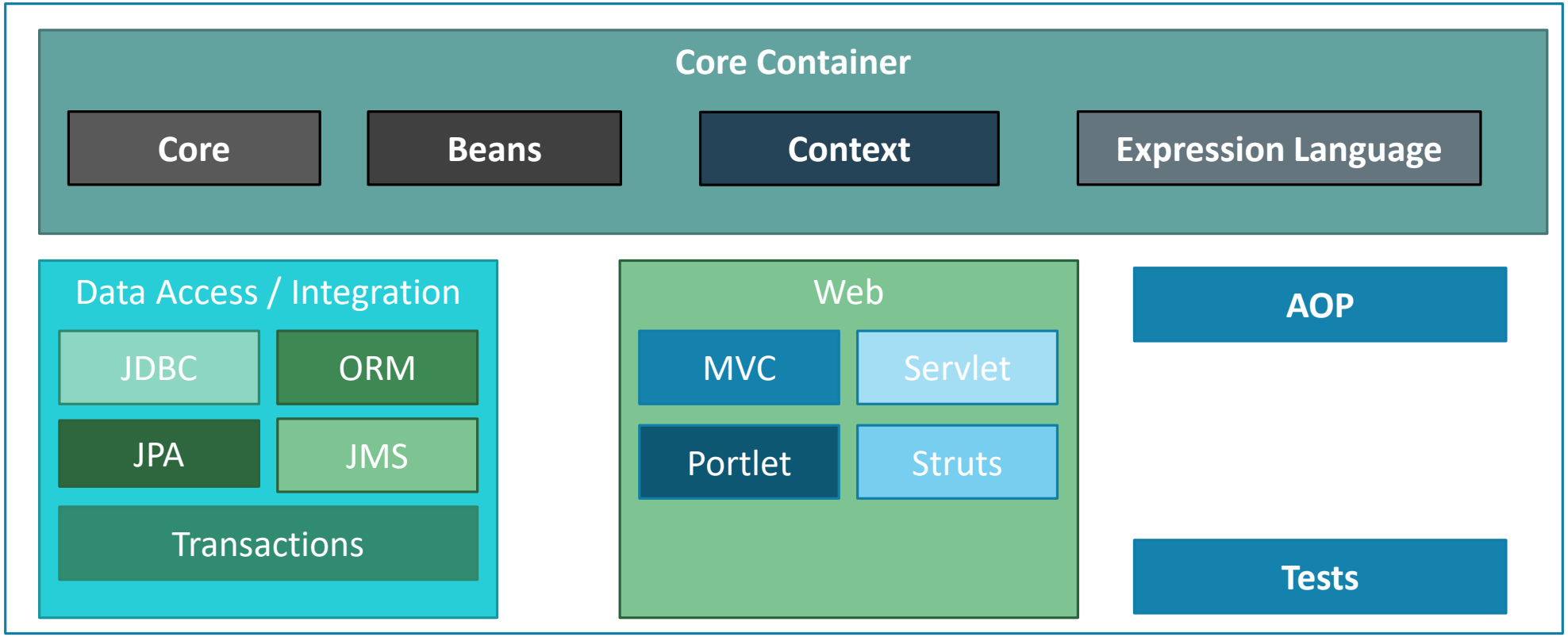
Spring gère des *JavaBean*, appelés *beans*

Fournit les mécanismes

- De fabrique d'objets (**BeanFactory**)
- D'inversion de contrôle (IoC – Inversion of Control), d'injection de dépendances

PRÉSENTATION DE SPRING

Constitués de modules



PRÉSENTATION DE SPRING

Dépendance Maven

- Spring-Context

Maven chargera les dépendances du Core de Spring

INJECTION DE DÉPENDANCES

Dependency Injection (DI)

RAPPEL : On parle de dépendance entre objets lorsque

- A a un attribut de type B
- A est de type B, ou implémente B
- A dépend d'un type C qui lui-même dépend d'un type B
- Une méthode de A appelle une méthode de B

Les interfaces permettent de nous abstraire de l'implémentation finale

- Mais nécessite toujours l'instanciation d'une classe concrète, l'instanciation de l'implémentation

INJECTION DE DÉPENDANCES

Soit les classes suivantes

```java
public interface IMusicien {
  public void jouer();
}
```

```java
public interface IInstrument { }
```

```java
public class Guitariste implements IMusicien {
  private IInstrument instrument = new Guitare();

  @Override public void jouer() {
    System.out.println("Le musicien joue : " + this.instrument);
  }
}
```

```java
public class Guitare implements IInstrument {
  @Override public String toString() {
    return "GLINK GLINK GLINK";
  }
}
```

- C’est fonctionnel, mais ce n’est pas bien parce que ce n’est pas son rôle d’instancier l’instrument !

INJECTION DE DÉPENDANCES

Dans un programme principal

```
public class Programme {
  public static void main(String[] args) {
    IMusicien myMusicien = new Guitariste();
    myMusicien.jouer();
  }
}
```

- C’est fonctionnel, mais ce n’est pas bien parce que ce n’est pas son rôle d’instancier le musicien !

INJECTION DE DÉPENDANCES

Créons une Factory qui se chargera

- D'instancier l'instrument

```
public class InstrumentFactory {
  public static IInstrument getInstrument() {
    return new Guitare();
  }
}
```

INJECTION DE DÉPENDANCES

Créons une Factory qui se chargera

- D'instancier le musicien

```
public class MusicienFactory {
  public static IMusicien getMusicien() {
    return new Guitariste(InstrumentFactory.getInstrument());
  }
}
```

- *Remarque : on va donner au musicien sa dépendance à l'instrument, en le transmettant par son constructeur (on aurait eu une classe abstraite, on aurait pu le gérer au niveau supérieur)*

```
public class Guitariste implements IMusicien {
  private IInstrument instrument;

  public Guitariste(IInstrument instrument) {
    this.instrument = instrument;
  }
}
```

INJECTION DE DÉPENDANCES

Dans notre programme principal, nous avions ceci

```
IMusicien myMusicien = new Guitariste();
```

Remplacé par ça

```
IMusicien myMusicien = MusicienFactory.getMusicien();
```

- La classe n'instancie plus, mais récupère sa dépendance via la Factory !

INJECTION DE DÉPENDANCES

Si nous voulons changer les implémentations, seules les Factories sont à modifier

Les classes Musicien et Programme ne gèrent plus l'instanciation

- C'est ce qu'on appelle l'injection de dépendances ou « Inversion of Control » (IoC)

C'est un travail fastidieux, en partie possible grâce à l'abstraction (interfaces)

- C'est là que SPRING entre en jeu !

INJECTION DE DÉPENDANCES

Spring s’appuie sur le pattern *Inversion of Control* (IoC)

Permet de rendre indépendantes les couches techniques

- IoC se charge d’instancier et de donner la référence créée !

INJECTION DE DÉPENDANCES

L'injection de dépendances n'est possible que dans le contexte de Spring Container

- Les objets y ont accès par un quelconque moyen
- Les objets sont déjà dans le contexte de Spring

Spring ne peut nous injecter que des objets qu'il manage

Spring injectera les dépendances **après** l'instanciation des objets

- Donc les références injectées ne sont pas disponibles dans le constructeur de l'objet

CONFIGURATION

Déclaration d'un *bean* par XML

- Fichier de configuration « context »
  - <bean />

Fichier "application-context.xml" à placer dans main/resources

Sous Eclipse

- Plugin Spring IDE (ou Spring Tools)
- Aide à la configuration Spring

DÉCLARER UN BEAN

Bean nommé **guitariste**, de type *fr.formation.musicien.Guitariste*

```
<bean id="guitariste" class="fr.formation.musicien.Guitariste" />
```

INJECTER UNE DÉPENDANCE

Injection de la référence « guitare » dans la propriété « instrument »

```
<bean id="guitare" class="fr.formation.instrument.Guitare" />

<bean id="guitariste" class="fr.formation.musicien.Guitariste">
  <property name="instrument" ref="guitare" />
</bean>
```

UTILISATION DEPUIS UNE CLASSE JAVA

Charger le conteneur Spring (XML) depuis une classe (Programme principal)

```
ClassPathXmlApplicationContext myContext = new ClassPathXmlApplicationContext("classpath:application-context.xml");
```

Récupérer un bean depuis le contexte Spring

- Par son nom

```
IMusicien myMusicien = (IMusicien)myContext.getBean("guitariste");
```

- Par son type

```
IMusicien myMusicien = myContext.getBean(IMusicien.class);
```

INJECTION DE DÉPENDANCES

En résumé

- On déclare en tant que bean les dépendances par une approche déclarative (XML ou annotations)
- Toutes nos déclarations de dépendances sont au même endroit

EXERCICE

Installer Spring IDE (ou Spring Tools)

Créer un nouveau projet (Maven)

Créer une classe principale « ProgrammeSpring » avec sa méthode *main*

Créer le modèle vu précédemment (Guitariste, Guitare et les interfaces)

Configurer Spring dans l'application

- application-context.xml
- (s'aider de http://formations.ascadis.fr/ajc/spring/01__configuration-spring.html)

Faire jouer le guitariste !

Ne faire aucune instanciation!

INJECTION DE DÉPENDANCES

Entre *beans* managés par Spring **uniquement**

Déclaration d'un *bean*

- Annotations
  - @Component
  - @Controller / @RestController
  - @Service
  - @Repository

Injection d'un *bean*

- @Autowired (avec @Qualifier("") possible)
- @Inject
- @Resource("")

LES ANNOTATIONS

Annoter les classes

Annotation	Définition	Cas d'utilisation
@Component	Composant Spring	Tous les cas, sauf … (voir ci-dessous)
@Controller	Composant de type Controller	Point d'accès (comme les Servlets)
@RestController	Composant de type Controller	Point d'accès Service Web REST
@Repository	Composant de type Repository	Classe entrepôt (DAO par exemple)
@Service	Composant de type Service	Fournisseur de service

LES ANNOTATIONS

Pour injecter une référence gérée par Spring

- Utilisation de l'annotation *@Autowired* sur une propriété
- Pour que ce soit fonctionnel, il faut que toutes les références soient gérées par Spring

```
@Component
public class Guitariste implements IMusicien {
  @Autowired
  private IInstrument instrument;
}
```

- Ne fonctionnera que s'il y a un *IInstrument*
  - Annoté de **@Component**
  - Ou déclaré en tant que **bean** dans le fichier de configuration XML

LES ANNOTATIONS

Il faut que les classes annotées soient scannées par SPRING

- Préciser les packages à scanner pour cette configuration

```
<context:component-scan base-package="fr.formation.musicien, fr.formation.instrument" />
```

EXERCICE

Remplacer la déclaration des **beans** XML en déclaration par annotation

INJECTION DE DÉPENDANCES

RAPPEL

- Spring injectera les dépendances **après** l'instanciation des objets
- Donc les références injectées ne sont pas disponibles dans le constructeur de l'objet

INJECTION DE DÉPENDANCES

Au besoin, utiliser l'annotation @PostConstruct sur une méthode

- Elle s'exécutera **juste après** l'instanciation !

```
@Component public class Guitariste implements IMusicien {
  @Autowired private IInstrument instrument;

  public Guitariste() {
    this.instrument; //Pas dispo
  }

  @PostConstruct
  public void init() {
    this.instrument; //Disponible
  }
}
```

INJECTION DE DÉPENDANCES

Ou utiliser *@Autowired* sur le constructeur (avec un argument du type de la dépendance)

- Dans ce cas, Spring cherchera à injecter la dépendance dès la construction de l'objet

```
@Component public class Guitariste implements IMusicien {
  private IInstrument instrument;

  @Autowired
  public Guitariste(IInstrument instrument) {
    this.instrument = instrument;
  }
}
```

EXERCICE

Reprendre l'exercice sur les musiciens

Cette fois, le chef d'orchestre décide de quel instrument les musiciens jouent

- Les musiciens « à vent » peuvent jouer de tout instrument « à vent »
- Les musiciens « à cordes » peuvent jouer de tout instrument « à corde »

Utiliser l'annotation @Qualifier("nom") pour retrouver un bean par son nom

LES ANNOTATIONS

Il est possible de configurer Spring au travers d'une classe de configuration

- Annotée de *@Configuration*
- Les packages scannés sont précisés dans l'annotation *@ComponentScan*
- Les **beans** créés sont annotés de *@Bean*

```java
@Configuration
public class AppConfig {
  @Bean public IInstrument guitare() {
    return new Guitare();
  }

  @Bean public IMusicien guitariste() {
    return new Guitariste();
  }
}
```

```java
@Configuration
@ComponentScan({ "fr.formation.musicien", "fr.formation.instrument" })
public class AppConfig {

}
```

LES ANNOTATIONS

On peut charger cette configuration dans le programme principal

```
AnnotationConfigApplicationContext myContext = new AnnotationConfigApplicationContext(AppConfig.class);

IMusicien myMusicien = myContext.getBean(IMusicien.class);
```

Ou utiliser, dans le fichier de configuration XML

```
<context:annotation-config />

<bean id="config" class="fr.formation.config.AppConfig" />
```

LES ANNOTATIONS

On peut aussi inclure un fichier de configuration XML pour une *@Configuration*

- Annotation *@ImportResource*

```
@Configuration
@ImportResource("classpath:application-context.xml")
public class AppConfig {

}
```

EXERCICE

Remplacer toute la configuration XML par de la configuration par classe

- Création d'une classe *AppConfig*
- Déclaration par annotation (scan des packages)