Java 4A SAGI 20-21 Les annotations Spring (résumé)

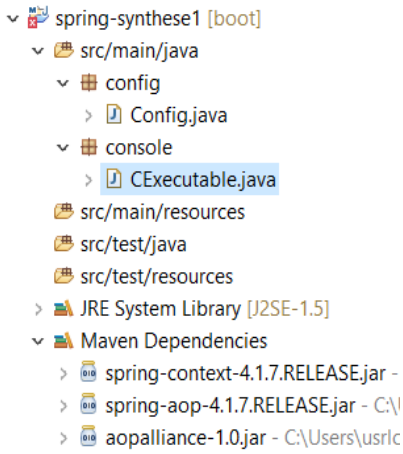
Spring est un framework qui permet, entre autres, d'instancier et relier les couches d'une application multicouches. Ce document résume les annotations Spring utiles dans ce cas. Un objet instancié par Spring est appelé "bean".

Pour utiliser Spring, on peut créer un projet *Maven* qui va se charger d'importer les librairies utiles. Ci-après, une configuration Maven (*pom.xml*) possible.

Un exemple minimal contient une classe exécutable, qui exploite le contexte Spring, et une classe de configuration qui définit les ***beans*** (objets créés par Spring).

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>istia.bc</groupId>
  <artifactId>spring-intro</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <dependencies>
        <dependency>
              <groupId>org.springframework</groupId>
              <artifactId>spring-context</artifactId>
              <version>4.1.7.RELEASE</version>
        </dependency>
  </dependencies>
</project>
```

Structure du projet éclipse




```
// config.java  [classe de configuration Spring]
package config;
// imports pour les annotations Spring
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration      // Annotation Spring
public class Config
{
       @Bean        // Annotation Spring
       public int varI()
       {
              return 137;
       }
}
```

Dans la classe annotée **`@Configuration`**, l'annotation **`@Bean`** fait de **varI** un ***bean***, c'est-à-dire un objet instancié par Spring.

Dans la fonction main(), après création du contexte Spring **ctx** (qui utilise la classe de configuration), on peut obtenir le bean nommé **`"varI".`**

```
package console;  // [CExecutable : classe de main()]
import config.Config;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class CExecutable
{
  public static void main(String[] args) {
       AnnotationConfigApplicationContext ctx;
       ctx = new AnnotationConfigApplicationContext(Config.class);
       try{
              int val = ctx.getBean("varI",Integer.class);
              System.out.println("val=" + val);  // sortie: val=137
       }
       catch(Exception ex){System.out.println(ex.getMessage());}
       ctx.close();
  }
}
```

Injection de paramètre : un second bean "valeur" est défini. Lors de sa définition, le paramètre `varI` (type int) correspond d'ailleurs au bean "varI". Un bean est défini à l'aide de la valeur d'un autre bean.

```java
package config;
import ...            // imports Spring
import entites_grp1.*;

@Configuration    // [configuration Spring]
public class Config
{
       @Bean
       public int varI(){ return 137; }

       @Bean
       public Valeur valeur(int varI)
       {  // varI = bean "varI"
          return new Valeur(varI);
       }
}
```

```java
// [classe Valeur]
package entites_grp1;

public class Valeur
{
       private int data;
       public Valeur(int d){
              setData(d); }

       public int getData() {
              return data;}

       public void setData(int data) {
              this.data = data;
       }
}
```

```java
package console;   // [classe executable]
import config.Config;
import ...    // imports Spring
import entites_grp1.*;

public class CExecutable {
       public static void main(String[] args) {
        AnnotationConfigApplicationContext ctx;
        ctx = new AnnotationConfigApplicationContext(Config.class);
        int val =ctx.getBean("varI",Integer.class);
        Valeur valeur = ctx.getBean("valeur",Valeur.class);
        System.out.println("val=" + val);                  // sortie: val=137
        System.out.println("valeur=" + valeur.getData()); // sortie: valeur=137
        ctx.close();
       }
}
```

@Component = crée un bean de la classe annotée (avec ou sans nom)
@ComponentScan = où chercher des bean créés par l'annotation @Component

```java
// [classe Personne]
package entites_grp1;
import org.springframework.stereotype.Component;

// un bean nommé "jean" de type Personne
@Component("jean")
public class Personne{
  private String nom;
  private int age;
  public Personne()
  { nom="Jean"; age=52;      }

  public String toString()
  {  return nom + ":" + age; }
}
```

```java
package config;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.ComponentScan;
import entites_grp1.*;

//cherche des beans dans package entites_grp1
@Configuration
@ComponentScan({"entites_grp1"})
public class Config{
       @Bean
       public int varI(){ return 137; }
}
```

```java
// [EXTRAIT de CExecutable.main(String[] args)]
       ...

  AnnotationConfigApplicationContext ctx;
  ctx = new AnnotationConfigApplicationContext(Config.class);
  Valeur valeur = ctx.getBean("valeur",Valeur.class);
  Personne p =ctx.getBean("jean",Personne.class); // bean "jean"
  System.out.println(p.toString());               // sortie: Jean:52
  ctx.close(); ...
```

@Autowired (injection de dépendance) : indique que la référence est initialisée pour pointer un bean Spring. Ceci permet de relier des beans (@Component) entre eux.

```java
// [classe Service]
package entites_grp1;
import org.springframework.stereotype.Component;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;

@Component            // crée un bean anonyme
public class Service {
        private String nom;

        @Autowired            // [injection de dépendance]
        @Qualifier("jean")    // chef<-bean nommé "jean"
        private Personne chef;

        public Service(){nom="compta";}

        public String toString(){
                return "Service:" + nom + ", resp=" + chef.toString();
        }
}
```

```java
// [Config. Spring]
package config;
import ...;  // imports Spring
import entites_grp1.*;

@Configuration
@ComponentScan({"entites_grp1"})
public class Config
{
        ...
}
```

```java
package console;  //[CExecutable.main()]
import entites_grp1.*;
import ...; // imports Spring
        ...

  AnnotationConfigApplicationContext ctx;
  ctx = new AnnotationConfigApplicationContext(Config.class);
  Service s =ctx.getBean(Service.class);   // bean Service anonyme...
  System.out.println(s.toString());        // ... lié au bean "jean"
  ctx.close(); ...
```

Sortie console

```
Service:compta, resp=Jean:52
```

Autre exemple : ici, le seul bean implémentant l'interface **IHello** est le bean anonyme **Espagnol**

```java
package entites_grp2; //[interface IHello]
public interface IHello
{
        public String hello();
}
```

```java
package config;        //[Configuration Spring]
...
// des beans @Component dans plusieurs packages
@Configuration
@ComponentScan({"entites_grp1","entites_grp2"})
public class Config
{       ...
}
```

```java
package entites_grp2; //[classe Espagnol]
import org.springframework.stereotype.Component;

@Component  // bean anonyme
public class Espagnol implements IHello
{
        public String hello() {
                return "Hola, que tal?";
        }
}
```

```java
package entites_grp2;      //[classe Reception]
import org.springframework.stereotype.Component;
import org.springframework.beans.factory.annotation.Autowired;

@Component            // bean anonyme
public class Reception {
  @Autowired
  IHello groom;   // groom<- bean Espagnol

  public String Accueil()  {
       return groom.hello();
  }
}
```

```java
package console;  // [classe exécutable]
import entites_grp1.*;
import entites_grp2.*;
import ...; // imports Spring
        ...
  AnnotationConfigApplicationContext ctx;
  ctx = new AnnotationConfigApplicationContext(Config.class);
  IHello ih =ctx.getBean(IHello.class);
  Reception rec =ctx.getBean(Reception.class);
  System.out.println(ih.toString());       // bean anonyme Espagnol
  System.out.println(rec.Accueil());       // bean Reception lié au bean Espagnol
  ctx.close(); ...
```

Sortie console

```
entites_grp2.Espagnol@1a1c291
Hola, que tal?
```

Illustration sur application multicouches Metier/Dao

```
// interface IMetier
package couches;

public interface IMetier{
        ...
}
```

```
// interface IDao
package couches;

public interface IDao{
  ...
}
```

```
// implémentation couche Metier (Metier1)
package couches;

import org.springframework.stereotype.Component;
import org.springframework.beans.factory.annotation.Autowired;

@Component
public class Metier1 implements IMetier
{
        @Autowired
        IDao dao;       //injection couche Dao

        @Override
        public String toString(){
                return "metier=" + super.toString()
                 + "dao=" + dao.toString();
        }
}
```

```
// implémentation couche Dao (Dao1)
package couches;
import org.springframework.stereotype.Component;

@Component
public class Dao1 implements IDao{
        ...
}
```

```
// [Configuration Spring]
package config;
import org.springframework.context.annotation.ComponentScan;

// couches = package des classes annotées @Component
@ComponentScan({"couches"})
public class ConfigCouches
{   ....    //autres beans @Bean si nécessaire
}
```

```
// classe exécutable : création
// contexte Spring et accès
// au bean métier

package console;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import config.ConfigCouches;  // import classe de configuration
import couches.IMetier;       // IMetier (seul type utilisé ici)

public class CExeCouches {

        public static void main(String[] args)
        {
         AnnotationConfigApplicationContext ctx;
         // création contexte Spring (qui crée les beans)
         ctx = new AnnotationConfigApplicationContext(ConfigCouches.class);
         IMetier metier = ctx.getBean(IMetier.class); //metier<-bean Metier1

         System.out.println(metier.toString());
               // la sortie console dit:    couche métier : bean Metier1
               //                           couche Dao : bean Dao1
         ctx.close();
        }
}       //  LES TYPES REELS (Metier1 et Dao1) des IMPLEMENTATIONS
        //  N'APPARAISSENT PAS ICI !!!
```

Sortie console

```
metier=couches.Metier1@16b4233 dao=couches.Dao1@1d66ad3
```

Alternative pour Metier/Dao : on peut aussi définir les beans des couches (dao et metier) dans la classe de configuration Spring; ces classes n'ont plus à être annotées `@Component`. S'il fallait changer l'implémentation des couches, il suffirait alors d'ajouter des classes implémentant `IDao` et `IMetier` et de changer la classe de configuration. Noter que l'injection de dépendance opère néanmoins.

```
// [Configuration Spring (alternative)]
package config;
import couches.Dao2;
import couches.IDao;
import couches.IMetier;
import couches.Metier2;

@Configuration
public class ConfigBis
{
  @Bean IDao dao(){ return new Dao2(); }

  @Bean IMetier metier(){ return new Metier2(); }
}
```

```
// interface IDao
package couches;

public interface IDao{
   ...
}
```

```
// interface IMetier
package couches;

public interface IMetier{
        ...
}
```

```
// couche Dao (Dao2)
package couches;
    // pas d'annotation @Component
public class Dao2 implements IDao{
        ...
}
```

```
// couche Metier (Metier2)
package couches;

import org.springframework.beans.factory.annotation.Autowired;

    // pas d'annotation @Component
public class Metier2 implements IMetier{
  @Autowired
  IDao dao;    // injection couche Dao

  @Override
  public String toString(){
       return "metier=" + super.toString()
        + "dao=" + dao.toString();
  }
}
```

```
// classe exécutable : création

package console;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import config.ConfigBis;
import couches.IMetier;

public class CExeCouches {

  public static void main(String[] args) {
       AnnotationConfigApplicationContext ctx;
       ctx = new AnnotationConfigApplicationContext(ConfigBis.class);
       IMetier metier = ctx.getBean(IMetier.class);
       System.out.println(metier.toString());
       ctx.close();
  }
}
```

Sortie console

```
metier=couches.Metier2@7e9a5fbedao=couches.Dao2@44a3ec6b
```