

Rappel Séance 2

(week-end 13-14/11/2021)

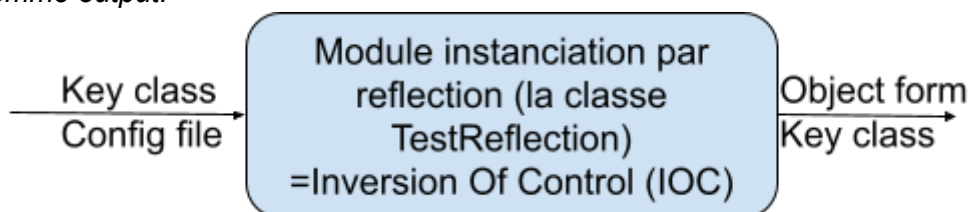
1. Dans votre projet "TPs Java" créer un package p1 = "ma.education.tp2.reflection.rappel"
2. Dans le package p1, créer une classe ConnectionDB définie par l'url (String), le user (String) et le password (String)
3. Dans le package p1, créer un fichier de configuration param.properties contenant :
keyClass1=ma.education.tp2.reflection.rappel.ConnectionDB
4. Dans le package p1, créer une classe TestReflection contenant la méthode main suivante:

```
public static void main(String[] args) {
    // Lecture fichier des paramètres
    ResourceBundle rb =
        ResourceBundle.getBundle("ma.education.tp2.reflection.rappel.param");
    String value = rb.getString("keyClass1");
    // Convert String type To Class Type
    Class c = Class.forName(value);
    // Tableau pour les constructeurs de la classe ConnectionDB
    Constructor[] constructors = c.getDeclaredConstructors();
    // Modification de la visibilité du premier constructeur
    constructors[0].setAccessible(true);
    // Instanciation de l'Objet (o) par réflexion.
    Object o = constructors[0].newInstance(null);
}
```

5. Mettre un affichage dans le constructeur de la classe ConnectionDB puis exécuter la classe TestReflection



Noter que la classe main prend un fichier de paramètre comme input et il crée un objet comme output.



Plusieurs frameworks java assure l'instanciation par reflection notamment **Spring IOC** que nous allons détailler dans la séance d'aujourd'hui.

Séance 3

(week-end 20-21/11/2021)

Les objectifs de la séance d'aujourd'hui:

Objectif 2.4 : Utiliser le mot réservé **"static"** (obj de la séance 2)

Objectif 2.5 : Comprendre le patron de conception : **Singleton**. (obj de la séance 2)

Objectif 3.1 : Utiliser Maven pour la gestion des dépendances d'un projet

Objectif 3.2 : Utiliser le Framework Spring IOC pour la création des objets (beans)

Objectif 3.3 : Assimiler les annotations et utiliser Lombok pour les getters et setters

Complément Youtube de cette séance : <https://youtu.be/kgBEOUFJ2us>

I. Le mot réservé **"static"**

Le mot clé static en Java est un modificateur utilisé pour économiser l'espace mémoire. Cela aide à gérer la mémoire occupée par les objets et leurs attributs. Le mot-clé static garantit qu'un seul espace mémoire sera créé pour représenter l'attribut déclaré static.

1. Créer un nouveau package p3= "ma.education.tp2.statickeyword"
2. Créer une nouvelle classe Etudiant avec les attributs id(long), nom (String) et nbEtudiants (int).

Pour le moment laisser la visibilité des attributs de la classe Etudiant "public".

3. D'abord Comparer les attributs de la classe Etudiant en donnant les réponses aux questions suivantes:

- a. Est ce que chaque étudiant a son propre Id? : Oui ou non
- b. Est ce que chaque étudiant a son propre nom? : Oui ou non
- c. Est ce que chaque étudiant a son propre nb Etudiant? : Oui ou non

Si la réponse est "oui" alors l'attribut est non static

Si la réponse est "non" alors l'attribut doit être static.

4. Déclarer l'attribut nbEtudiants en utilisant le mot réservé static
5. Créer un constructeur pour les trois attributs comme suivant.

```
public Etudiant (long id, String nom,int nb) {  
    this.id = id;  
    this.nom = nom;  
    nbEtudiants+=nb;  
}
```

6. Dans le package p3, créer la classe TestStatic contenant la méthode main.
7. Dans la méthode main, instancier les deux objets suivants:
Etudiant e1= new Etudiant(1, "AHMED", 20);
Etudiant e2= new Etudiant(2, "SAID", 30);
8. Afficher les attributs de (e1) et (e2)
9. Que dites vous par rapport à la valeur de l'attribut **static** nbEtudiants?



NB : Étant donné que les variables statiques appartiennent à une classe, elles seront accessibles directement à l'aide du nom de la classe et ne nécessitent aucune référence d'objet.

Les variables statiques ne peuvent être déclarées qu'au niveau de la classe

Les variables statiques sont accessibles sans création d'objet

Une méthode statique ne peut utiliser « this. » et « super. » dans son corps.

Une méthode statique ne peut accéder ni aux variables non statiques ni aux méthodes non statiques

II. Le patron de conception “Singleton”

Le singleton est un patron de conception (design pattern) dont l'objectif est de restreindre l'instanciation d'une classe à un seul objet (ou bien à quelques objets seulement). Il est utilisé lorsqu'on a besoin exactement d'un objet pour coordonner des opérations dans un système. Le modèle est parfois utilisé pour son efficacité, lorsque le système est plus rapide ou occupe moins de mémoire avec peu d'objets qu'avec beaucoup d'objets similaires.

1. Créer un nouveau package p4= “ma.education.tp4.singleton”
2. Créer une classe Terre définie par les attributs distanceToSoleil (double) et surface (double)
3. Vu qu'il n'y a qu'une seule Terre, cette classe doit créer un seul objet dans la mémoire. Pour le faire, on doit assurer les trois règles suivantes
 - a. Verrouiller le constructeur de cette classe en utilisant la visibilité “private”


```
private Terre(double distanceToSoleil, double surface) {
    this.distanceToSoleil = distanceToSoleil;
    this.surface = surface;
}
```
 - b. Créer un attribut private static ayant le type Terre et le nom instance


```
private static Terre instance;
```
 - c. Créer une méthode public et static ayant le type de retour Terre et le nom getInstance

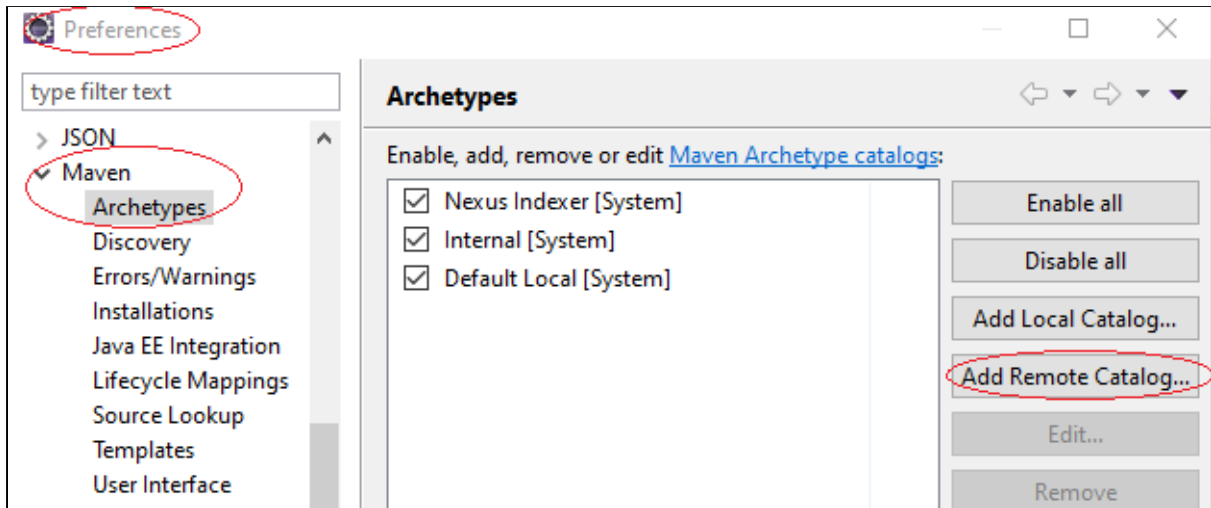

```
public static Terre getInstance(double distanceToSoleil, double surface) {
    if (instance == null)
        instance = new Terre(distanceToSoleil, surface);
    return instance;
}
```
 - d. Créer une classe TestSingleton avec la méthode main:
 - e. Dans la méthode main créer les deux objets suivants.


```
Terre t1= Terre.getInstance(1000, 2000);
Terre t2= Terre.getInstance(4000, 5000);
```
 - f. Afficher la distanceToSoleil et la surface pour l'objet (t1)
 - g. Afficher la distanceToSoleil et la surface pour l'objet (t2). Quelle est votre remarque par rapport au nombre d'objets créés dans la mémoire.

III. Création d'un projet Maven

1. Ajoutez le catalogue Maven distant. [Il faut être connecté à Internet]

a. Windows -> Preferences -> Maven -> Archetypes->

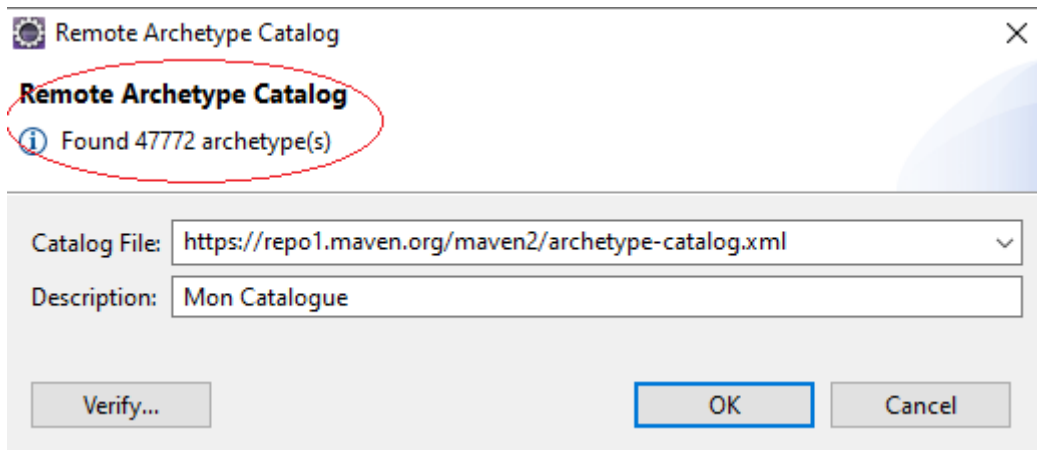


b. Cliquez sur le bouton Add Remote Catalog.

Saisir le lien du fichier catalogue :

<https://repo1.maven.org/maven2/archetype-catalog.xml>

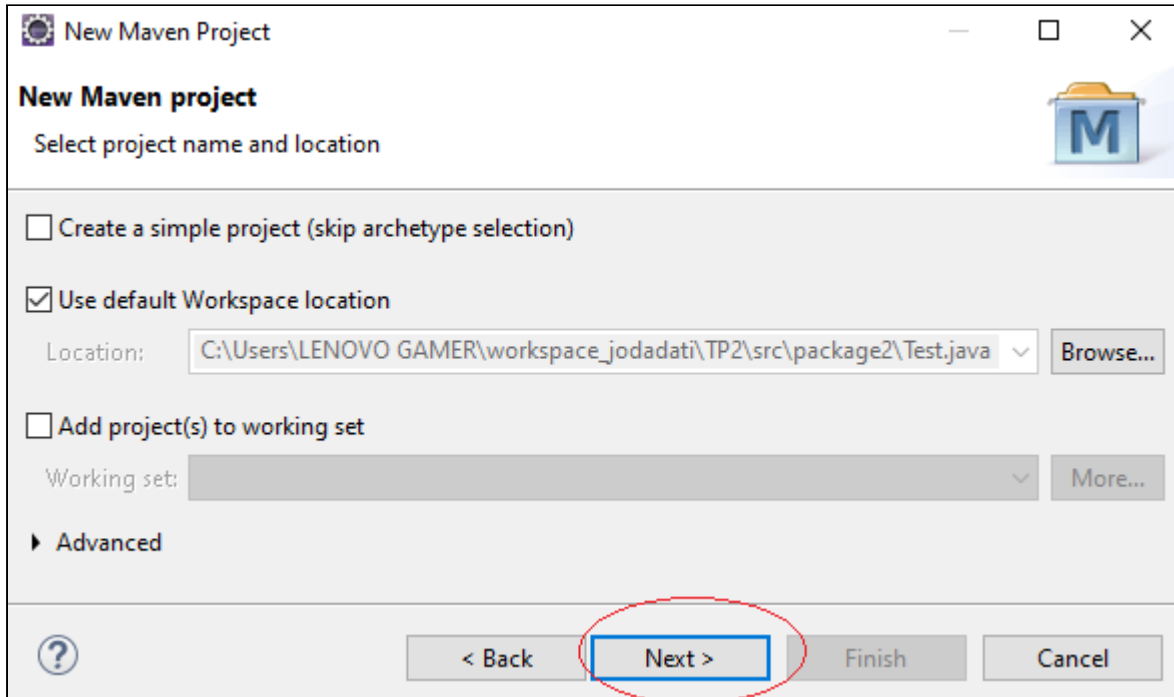
Ajouter une description de votre choix, "Mon Catalogue" à titre d'exemple



c. Cliquez sur le bouton Verify pour voir le nombre des types d'architecture trouvées.

2. Maintenant créer un projet Maven

File->New->Project->Maven->MavenProject



New Maven Project

Select project name and location

☐ Create a simple project (skip archetype selection)

☒ Use default Workspace location

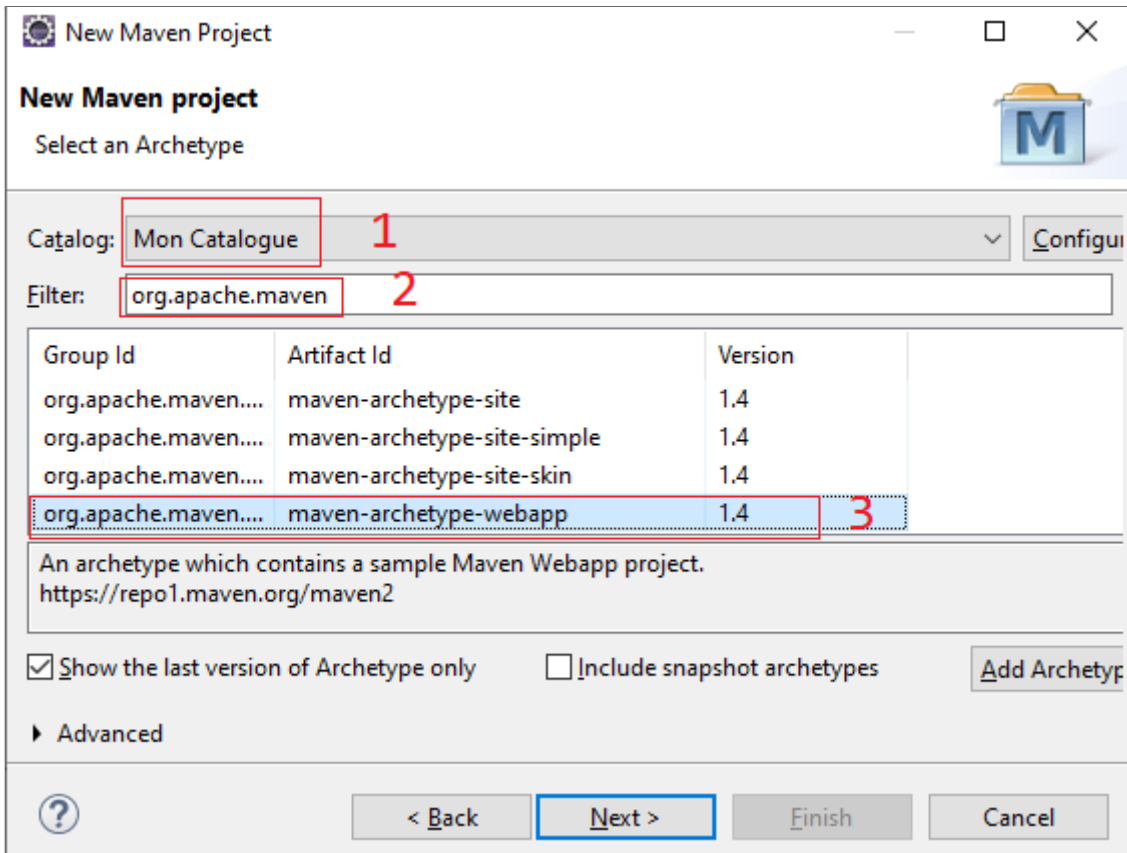
Location:

☐ Add project(s) to working set

Working set:

► Advanced

Cliquer sur "Next"



New Maven Project

Select an Archetype

Catalog:

Filter:

Group Id	Artifact Id	Version
org.apache.maven....	maven-archetype-site	1.4
org.apache.maven....	maven-archetype-site-simple	1.4
org.apache.maven....	maven-archetype-site-skin	1.4
org.apache.maven....	maven-archetype-webapp	1.4

An archetype which contains a sample Maven Webapp project.
<https://repo1.maven.org/maven2>

☒ Show the last version of Archetype only ☐ Include snapshot archetypes

► Advanced

- Choisir le catalogue que vous avez créé dans l'étape (1) . "Mon catalogue" dans mon cas.
- Filtrer les architectures du catalogue par "org.apache.maven"
- Choisir l'artifactId = **maven-archetype-webapp** version **1.4**
- Cliquer sur "Next"

New Maven project
Specify Archetype parameters

Group Id:

Artifact Id:

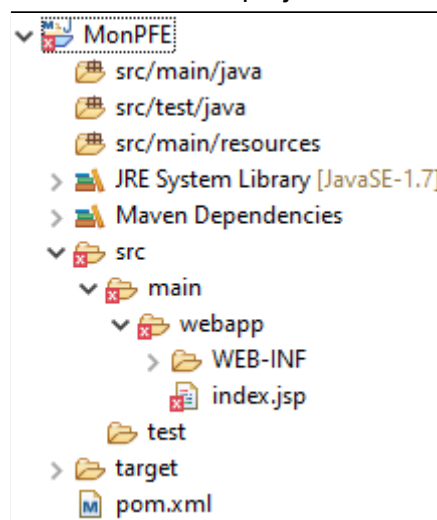
Version:

Package:

Properties available from archetype:

Name	Value

- Dans le "Group Id" saisir le nom du Package "ma.education.tp3"
- Dans "ArtifactId" saisir le nom de votre projet PFE. "MonPFE"
- Cliquer sur Finish et vérifier la structure du projet Maven suivante:



- Supprimer "index.jsp" qui présente une erreur de compilation.

- i. Remarque que les composants de base d'un projet maven sont : JRE System Library, **Maven Dependencies**, src,target et **pom.xml**.

Le package "src" :

Dans "src", on trouve un dossier "main" qui contient un autre répertoire "java" : c'est l'emplacement de nos classes java.

On peut aussi voir qu'il y a un répertoire "resources" qui va contenir les images et la configuration nécessaire pour notre projet.

Le fichier "pom.xml" :

Ce fichier xml permettra d'ajouter des bibliothèques à notre projet (dependencies)

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

A titre d'exemple les balises ci-dessus vont ajouter les jars suivants:

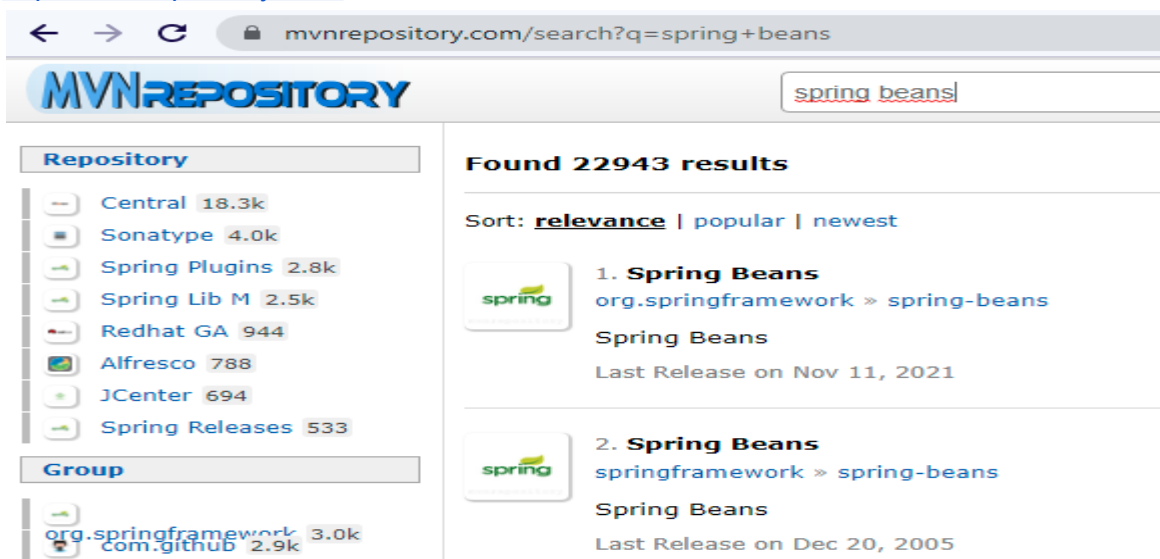
```

Maven Dependencies
> junit-4.11.jar - C:\Users\LENOVC
> hamcrest-core-1.3.jar - C:\Users\
```

Le pom.xml permet aussi de faire le build du projet pour le déployer en production.

IV. Ajout du Spring beans dependencies


Pour chercher une bibliothèque nécessaire pour votre projet, il faut aller à <https://mvnrepository.com/>



The screenshot shows the Maven Repository search results for "spring beans". The search bar contains "spring beans" and the results are sorted by "relevance". The first result is "Spring Beans" by "org.springframework", with a last release on Nov 11, 2021. The second result is also "Spring Beans" by "springframework", with a last release on Dec 20, 2005. On the left, there is a sidebar with "Repository" and "Group" sections. The "Repository" section lists various repositories like Central, Sonatype, Spring Plugins, etc. The "Group" section lists groups like org.springframework, com.github, etc.

1. Dans notre cas, nous avons cherché "Spring beans dependency"

Home » org.springframework » spring-beans



Spring Beans

Spring Beans

License Apache 2.0

Categories Dependency Injection

Tags beans spring dependency-injection

Used By 5,937 artifacts

Central (210) | Atlassian 3rd-P Old (1) | Spring Plugins (51) | Spring Lib M (1) | Spring Milestones (6) | JBoss Public (4) | ICM (3) | Grails Core (6) | Geomajas (1) | EBIPublic (4) | Alfresco (9) | Cambridge (1) | Gradle Releases (1)

Version	Repository	Usages	Date
5.3.13	Central	43	Nov, 2021
5.3.12	Central	472	Oct, 2021
5.3.11	Central	57	Oct, 2021
5.3.10	Central	467	Sep, 2021
5.3.9	Central	616	Jul, 2021
5.3.8	Central	591	Jun, 2021

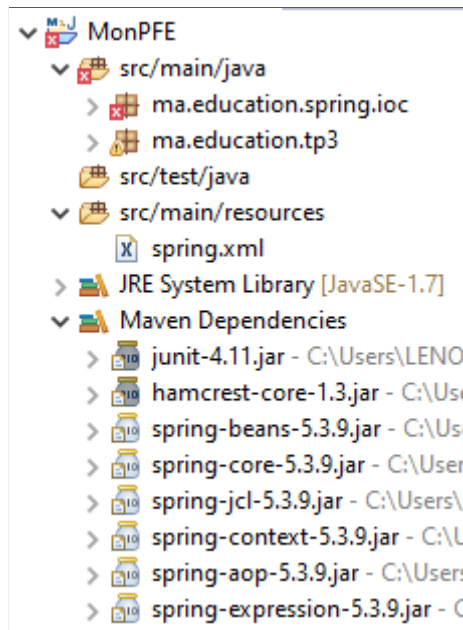
- Cliquer sur la version Spring Beans 5.3.9 et copier ses balises dependency dans le pom.xml de votre projet. Les dépendances de votre pom.xml deviennent:

```

<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-beans</artifactId>
    <version>5.3.9</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.3.9</version>
  </dependency>
</dependencies>

```

Sauvegarder le pom.xml. Puis vérifier "Maven dependencies" de votre projet



V. IOC par le conteneur ApplicationContext

3. Dans "src/main/java", créer un package p2 = "ma.education.tp3"
4. Dans p2, créer la classe Client suivante :

```
public class Client {
    private long id;
    private String name;
    public Client() {
        System.out.println("Acces Constructor Client..");
    }
}
```

5. Dans "src/main/resources", créer le fichier de configuration "spring.xml" suivant

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/util
        http://www.springframework.org/schema/util/spring-util.xsd">

    <bean id="clt" class="ma.education.tp3.Client"></bean>

</beans>
```

6. Dans "src/main/java", créer un package p3 = "ma.education.spring.ioc"
7. Dans p3 créer une classe TestSpringlocApplicationContext contenant la méthode main.

```
package ma.education.spring.ioc;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;
import ma.education.tp3.Client;

public class TestSpringIocApplicationContext{

    public static void main(String[] args) {
        ApplicationContext appContext=new
        ClassPathXmlApplicationContext("spring.xml");
        Client c = (Client ) appContext.getBean("clt");
    }
}
```

8. Exécuter la classe TestSpringIocApplicationContext et vérifier que le constructeur sans paramètres de la classe Client a été appelé par ApplicationContext de Spring IOC. Vous aurez l'affichage "**Access Constructor Client**"
9. A quelle instruction le constructeur de la Client est appelé ? dans (1) ou (2)
 - (1) ApplicationContext appContext=new ClassPathXmlApplicationContext("spring.xml");
 - (2) Client c = (Client) appContext.getBean("clt");
 Mettre (2) en commentaire pour vérifier.
10. Maintenant essayez de créer deux objets de la classe Client c1 et c2.


```
Client c1 = (Client ) appContext.getBean("clt");
Client c2 = (Client ) appContext.getBean("clt");
```



Questions :

- Combien de fois le constructeur de la classe Client sera appelé?
- Si le constructeur est déclaré private, sera t il appelé par l'ApplicationContext? Mettre le constructeur private et exécuté la classe TestSpringIoc



NB:

- Le scope des objets de la classe Client est par défaut "Singleton"
- ```
<bean id="clt" class="ma.education.tp3.Client" scope="singleton"></bean>
```

#### Le conteneur **ApplicationContext**:

- Instancier tous les objets **singleton** au parsing du fichier de configuration. Cette instantiation se fait une seule fois même si le développeur a appelé getBean plusieurs fois.
- Instancier tous les objets **prototype** lors de l'appel de la méthode getBean("clt"). Le nombre d'objets créés est équivalent au nombre d'appels de la méthode getBean.

## VI. IOC par le conteneur BeanFactory

11. Créer une classe `TestSpringIocBeanFactory` contenant la méthode main suivante:

```
package ma.education.spring.ioc;

import
org.springframework.beans.factory.support.DefaultListableBeanFactory;
import org.springframework.beans.factory.xml.XmlBeanDefinitionReader;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;

import ma.education.tp3.Client;

public class TestSpringIocBeanFactory {

 public static void main(String[] args) {
 final Resource resource = new ClassPathResource("spring.xml");
 final DefaultListableBeanFactory beanFactory = new
DefaultListableBeanFactory();
 final XmlBeanDefinitionReader xmlBeanDefinitionReader = new
XmlBeanDefinitionReader(beanFactory);
 xmlBeanDefinitionReader.loadBeanDefinitions(resource);
 Client client = (Client) beanFactory.getBean("clt");
 }
}
```



Le conteneur **BeanFactory**: Instancier les objets à la demande (lorsque la méthode `getBean("clt")` est appelée) et non pas au parsing du fichier de configuration.

Question : Quelle est la différence entre les conteneurs Spring : `ApplicationContext` et `BeanFactory`

## VII. Injection de dépendance par constructeur

1. Les objets `Client` créés précédemment portent des valeurs par défaut dans leurs attributs. Pour injecter des valeurs dans leurs attributs, il faut créer un constructeur avec paramètres et changer le fichier de configuration `spring.xml`.

- Le constructeur de la classe `Client`:

```
public Client(long id, String name) {
 this.id = id;
 this.name = name;
}
```

- Le fichier de configuration `spring.xml`:

```
<bean id="clt" class="ma.education.tp3.Client" scope="singleton">
 <constructor-arg value="1"/>
 <constructor-arg value="client1"/>
</bean>
```

- La classe de TestSpringIocApplicationContext

```
public class TestSpringIocApplicationContext {
 public static void main(String[] args) {
 ApplicationContext appContext=new
 ClassPathXmlApplicationContext("spring.xml");
 Client c1 = (Client) appContext.getBean("clt");
 System.out.println(c1.id);
 System.out.println(c1.name);
 }
}
```

Executer la classe TestSpringIocApplicationContext .

*Dans le fichier de configuration, le tag <constructor-arg> permet de fournir un paramètre au constructeur. L'attribut value permet de fournir une valeur au paramètre. Si les types ne sont pas identifiables ou ne suffisent pas pour déterminer de façon certaine le constructeur à utiliser, le conteneur utilise l'ordre des paramètres définis dans la configuration. Pour faciliter encore plus le travail, l'attribut index permet de préciser l'index du paramètre définit. Le premier paramètre possède l'index 0. Cette injection de dépendance s'appelle injection par constructeur.*

## VIII. L'injection par les Setters

Dans ce cas, la classe doit contenir les setters et les getters en respectant les conventions JavaBean pour les accesseurs.

- Les getters et les setters de la Client:

```
package ma.education.tp3;

public class Client {
 public long id;
 public String name;
 public long getId() {
 return id;
 }
 public void setId(long id) {
 this.id = id;
 }
 public String getName() {
 return name;
 }
 public void setName(String name) {
 this.name = name;
 }
}
```

- Le fichier de configuration spring.xml:

```
<bean id="clt" class="ma.education.tp3.Client" scope="singleton">
 <property name="id" value="2"/>
 <property name="name" value="clt2"/>
</bean>
```

- La classe de TestSpringIocApplicationContext

```
public class TestSpringIocApplicationContext {
 public static void main(String[] args) {
 ApplicationContext appContext=new
 ClassPathXmlApplicationContext("spring.xml");
 Client c1 = (Client) appContext.getBean("clt");
 System.out.println(c1.id);
 System.out.println(c1.name);
 }
}
```

Executer la classe TestSpringIocApplicationContext .

Le conteneur va créer une instance en invoquant le constructeur par défaut puis va invoquer le setter de chaque dépendance (attributs) en lui passant en paramètre celle définie dans la configuration.

## IX. Annotations

1. Dans votre projets "Mes TPs Java", créer un package "ma.education.tp4"
2. Créer une annotation Programmer, les annotations sont créées en utilisant le mot réservé @interface . Cette annotation contient les deux signatures: int id(); et String name();

```
package ma.education.tp4;
public @interface Programmer {
 abstract int id();
 String name();
}
```

3. Cette annotation sera appliquée seulement aux classes et aux interfaces. Pour le dire, l'annotation Programmer doit être annoté par l'annotation @Target

```
@Target(ElementType.TYPE)
public @interface Programmer {
 abstract int id();
 String name();
}
```

*ElementType.TYPE = Class, interface (including annotation type), or enum declaration*

4. Dans ma.education.tp4, créer la classe Calculatrice, en utilisant l'annotation précédente @Programmer mentionner le programmeur qui a développé la classe Calculatrice.

```
@Programmer(id=10,name="Said ALAMI")
public class Calculatrice {
}
```

5. Créer une classe TestReflectionAnnotation pour afficher les valeurs de l'annotation @Programmer utilisées dans la classe Calculatrice

```
public class TestReflectionAnnotation {
 public static void main(String[] args) {
 Class c = Calculatrice.class;
 Programmer programmer = (Programmer)
```

```
c.getDeclaredAnnotation(Programmer.class);
 System.out.println(programmer.id()+" "+programmer.name());
 }
}
```

Exécuter cette classe et remarquer que l'annotation n'existe pas au moment de l'exécution  
**Exception in thread "main" java.lang.NullPointerException**



Une annotation est définie par sa rétention, c'est-à-dire la façon dont elle sera conservée. La rétention est définie grâce à la méta-annotation `@Retention`. Les différentes rétentions d'annotation possibles sont :

**SOURCE** : L'annotation est accessible durant la compilation mais n'est pas intégrée dans le fichier .class généré.

**CLASS** : L'annotation est accessible durant la compilation, elle est intégrée dans le fichier .class généré mais elle n'est pas chargée dans la JVM à l'exécution.

**RUNTIME** : L'annotation est accessible durant la compilation, elle est intégrée dans le fichier class généré et elle est chargée dans la JVM à l'exécution. Elle est accessible par introspection (la réflexion).

- Ajouter alors `@Retention(RetentionPolicy.RUNTIME)` à l'annotation `@Programmer` et exécuter encore une fois la classe `TestReflectionAnnotation`

```
package ma.education.tp4;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
public @interface Programmer {
 abstract int id();
 String name();
}
```

- Créer une classe fille de `Calculatrice` et appeler la `CalculatriceMath`. Est ce que la classe fille va hériter l'annotation `@Programmer` de sa classe mère `Calculatrice`.

```
public class CalculatriceMath extends Calculatrice{
}
```

- Changer la classe `TestReflectionAnnotation` pour vérifier si `CalculatriceMath` est aussi annotée par `@Programmer`

```
public class TestReflectionAnnotation {
 public static void main(String[] args) {
 Class c = CalculatriceMath.class;
 Programmer programmer = (Programmer)
c.getAnnotation(Programmer.class);
 System.out.println(programmer.id()+" "+programmer.name());
 }
}
```

```
}
}
```

N'oublier pas de changer `c.getDeclaredAnnotation(Programmer.class)` par `c.getAnnotation(Programmer.class)`

Exécuter cette classe et remarquer l'exception : `java.lang.NullPointerException`

9. Annoter l'annotation `@Programmer` par l'annotation `@Inherited` et refaire l'exécution de la classe `TestReflectionAnnotation`. C'est quoi votre remarque?

### Exercice :

Annoter le package "ma.education.tp4" par l'annotation `@Programmer` et afficher les valeurs de l'annotation en utilisant la réflexion.

## X. Application des annotations : Lombok dependency

1. Ajouter dans le `pom.xml` de votre projet Lombok dependency

```
<dependency>
 <groupId>org.projectlombok</groupId>
 <artifactId>lombok</artifactId>
 <version>1.18.22</version>
</dependency>
```

2. Dans la classe `Calculatrice`, utiliser les annotations Lombok pour créer les getters, les setters et les constructeurs:

```
package ma.education.tp4;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.Setter;
@Programmer(id=10,name="Said ALAMI")
@Getter
@Setter
@AllArgsConstructor
public class Calculatrice {
}
```