

Rappel Séance 2

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

1. Donne le résultat d'affichage suite à l'exécution du programme suivant:

```
public class Rappel1 {
    static int vitesse = 10;
    Rappel1(int v){
        this();
        vitesse+=v;
    }
    Rappel1(){
        vitesse++;
    }
    public static void main(String[] args) {
        Rappel1 r1=new Rappel1(20);
        Rappel1 r2=new Rappel1(30);
        System.out.println(r1.vitesse);
        System.out.println(r2.vitesse);
    }
}
```

2. Donner le résultat d'affichage du programme suivant:

```
public class Rappel2 {
}

public class RappelFille extends Rappel2 {
    int vitesse = 10;
    static void afficher() {
        System.out.println(vitesse);
    }
    public static void main(String[] args) {
        Rappel2 r = new RappelFille();
        r.afficher();
    }
}
```

Séance 3

(week-end 27-28/11/2021)

Les objectifs de la séance d'aujourd'hui:

Objectif 4.1 : Utilisation du mot réservé "**abstract**"

Objectif 4.2 : Création des **interfaces** en Java

Objectif 4.3 : Création des **annotations** en Java

Objectif 4.4 : Utilisation de Spring IOC : ApplicationContext et BeanFactory

Complément Youtube de cette séance : <https://youtu.be/EALn43Zglpc>

I. Le mot réservé “abstract”

Le mot-clé **abstract** est un modificateur, utilisé pour les classes et les méthodes.

Classe abstraite : Une classe abstraite est une classe restreinte qui ne peut pas être utilisée pour créer des objets, pour y accéder, elle doit être héritée d’une autre classe.

Méthode abstraite : Une méthode abstraite ne peut être utilisée que dans une classe abstraite et elle n’a pas de corps. Le corps est fourni par la classe fille.

Exemple :

- La méthode `calculerSurface()` ne peut pas avoir un corps dans la classe `FormeGeometrique` dont elle doit être déclarée abstraite.

1. Créer un nouveau package `p4= “ma.education.tp.abstarctkeyword”`
2. Créer une classe nouvelle classe ***FormeGeometrique*** avec l’attribut `surface(double)` et la méthode ***public double calculerSurface() {}***
3. Supprimer le corps `}` de la méthode `calculerSurface()` et le remplacer par point virgule ;
4. Remarquer l’erreur de compilation, déclarer alors la méthode `calculerSurface()` ***abstract public abstract double calculerSurface();***
5. Remarquer les erreurs de compilation au niveau de la classe ***FormeGeometrique***. Déclarer alors cette classe **abstract**.

```
package ma.education.tp.abstarctkeyword;
// la classe est déclarée abstract
public abstract class FormeGeometrique {

    double surface;
    public abstract double calculerSurface(); // pas de corps de méthode
}
```

6. Maintenant créer une classe Fille de la classe ***FormeGeometrique***. Appeler la ***FormeGeoTypeA*** Remarque que cette classe fille présente des erreurs de compilation et qu’elle doit soit:
 - a. Donner un corps à la méthode abstraite, `calculer Surface()` .
 - b. Ou bien Se déclarer abstraite
7. Déclarer alors la classe ***FormeGeoTypeA*** abstraite.

```
package ma.education.tp.abstarctkeyword;

public abstract class FormeGeoTypeA extends FormeGeometrique {
}
```

8. Maintenant, créer encore une classe fille de la classe ***FormeGeoTypeA*** avec le nom ***Triangle***. Remarque que la classe `Triangle` est dans l’obligation de :
 - a. Donner un corps à la méthode abstraite, `calculer Surface()` .
 - b. Ou bien Se déclarer abstraite

9. Créer une méthode concrète `afficherSurface()` au niveau de la classe `FormeGeometrique`.

```
package ma.education.tp.abstarctkeyword;
public abstract class FormeGeometrique {
    double surface;
    // Méthode abstraite sans corps
    public abstract double calculerSurface();
    // Méthode concrète avec un corps
    public void afficherSurface() {
        System.out.println(this.surface);
    }
}
```



Les classes abstraites ne sont pas à 100% abstraites. Elles peuvent contenir un mélange des méthodes abstraites et des méthodes concrètes.

10. Créer une classe `Test` avec les instanciations suivantes. Remarquer que `f1` et `f2` présentent des erreurs de compilations car les classes abstraites ne sont pas instanciables

```
package ma.education.tp.abstarctkeyword;
public class Test {
    public static void main(String[] args) {
        FormeGeometrique f1 = new FormeGeometrique();
        FormeGeometrique f2 = new FormeGeoTypeA();
        FormeGeometrique f3 = new Triangle();
        FormeGeometrique f4 = new FormeGeometrique() {
            @Override // Classe fille Anonyme
            public double calculerSurface() {
                return 22;
            }
        };
    }
}
```

Remarquer que les classes abstraites peuvent être instanciées soit par une classe fille concrète (`f3`) ou bien par une classe fille anonyme (`f4`)



Les classes abstraites sont très utilisées dans l'API standard et les frameworks. Le framework Spring en possède de nombreuses. On peut énoncer deux principes pour justifier la création d'une classe abstraite :

- La classe que l'on écrit va être étendue par plusieurs classes filles.
- Les classes filles doivent obligatoirement posséder un corps de la méthode déclarée abstraite dans la classe mère.

II. Création des interfaces en java

Une interface permet de définir un ensemble de services qu'un client peut obtenir d'un objet. Une interface introduit une abstraction pure qui permet un découplage maximal entre un service et son implémentation. On retrouve ainsi les interfaces au cœur de l'implémentation de beaucoup de bibliothèques et de frameworks. Le mécanisme des interfaces permet d'introduire également une forme simplifiée d'héritage multiple.

1. Créer un nouveau package p6= "ma.education.tp.interfacekeyword"
2. Créer une interface java avec le nom **Compte**

```
package ma.education.tp.interfacekeyword;  
public interface Compte {  
}
```

Comme pour une classe, une interface a une portée, un nom et un bloc de déclaration. Une interface est déclarée dans son propre fichier qui porte le même nom que l'interface. Pour l'exemple ci-dessus, le fichier doit s'appeler Compte.java.

Une interface décrit un ensemble de méthodes en fournissant uniquement leur signature.

3. Créer les signatures suivante dans l'interface compte

```
package ma.education.tp.interfacekeyword;  
public interface Compte {  
    void deposer(int montant) ; // Signature méthode 1  
    int retirer(int montant); // Signature méthode 2  
    int getBalance(); // Signature méthode 3  
}
```

Une interface introduit un nouveau type d'abstraction qui définit à travers ces méthodes un ensemble d'interactions autorisées. Une classe peut ensuite implémenter une ou plusieurs interfaces.



Les méthodes d'une interface sont par défaut publiques et abstraites. Il n'est pas possible de déclarer une autre visibilité que public.

```
public interface Mobile {  
    public abstract void deplacer();  
}
```

L'interface ci-dessus est strictement identique à celle-ci :

```
public interface Mobile {  
    public abstract void deplacer();  
}
```

III. Implémentation des interfaces en java

1. Créer une classe CompteBancaire qui implémente l'interface Compte précédente:

```
public class CompteBancaire implements Compte{
}
```

2. Remarquer les erreurs de compilation et donner une implémentation (un corps) aux signatures déclarées dans l'interface Compte

```
package ma.education.tp.interfacekeyword;
public class CompteBancaire implements Compte {
    private String numero;
    private int balance;
    public CompteBancaire(String numero) {
        this.numero = numero;
    }
    @Override
    public void depoter(int montant) {
        this.balance += montant;
    }
    @Override
    public int retirer(int montant) {
        if (balance < montant) {
            return 0;
        }
        return this.balance -= montant;
    }
    @Override
    public int getBalance() {
        return this.balance;
    }
    public String getNumero() {
        return numero;
    }
}
```



L'implémentation des méthodes d'une interface suit les mêmes règles que la redéfinition.

Si la classe qui implémente l'interface est une classe abstraite, alors elle n'est pas obligée de fournir une implémentation pour les méthodes de l'interface.

Même si les mécanismes des interfaces sont proches de ceux des classes abstraites, ces deux notions sont clairement distinctes. Une classe abstraite permet de mutualiser une implémentation dans une hiérarchie d'héritage en introduisant un type plus abstrait. Une interface permet de définir les interactions possibles entre un objet et ses clients. Une interface agit comme un contrat que les deux parties doivent remplir. Comme l'interface n'impose pas de s'insérer dans une hiérarchie d'héritage, il est relativement simple d'adapter une classe pour qu'elle implémente une interface.

3. Créer les deux interfaces Carnivore et Herbivore suivantes :

Dossier des travaux pratiques. Module 1 : Java de base .Années scolaire 2021/2022. Niveau : Licence FST Settat

Professeur : M. Boulchahoub Hassan hboulchahoub@gmail.com

Mise à jour 18 Nov 2021

```
package ma.education.tp.interfacekeyword;
public interface Carnivore {
    void manger(Animal animal);
}

package ma.education.tp.interfacekeyword;
public interface Herbivore {
    void manger(Vegetal vegetal);
}
```

4. Vu qu'une classe en java peut implémenter plusieurs interface, Créer la classe Humain suivante:

```
package ma.education.tp.interfacekeyword;

public class Humain extends Animal implements Carnivore, Herbivore
{
    @Override
    public void manger(Animal animal) {
        // ...
    }
    @Override
    public void manger(Vegetal vegetal) {
        // ...
    }
}
```

Dans l'exemple précédent, la classe Humain implémente les interfaces Carnivore et Herbivore.

IV. Création et utilisation des annotations en java

1. Dans votre projets "Mes TPs Java", créer un package "ma.education.tp.annotations"
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.tp.annotations;
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.tp.annotations`, 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).

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

```
package ma.education.tp.annotations;

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();
}
```

7. 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{
}
```

8. 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`

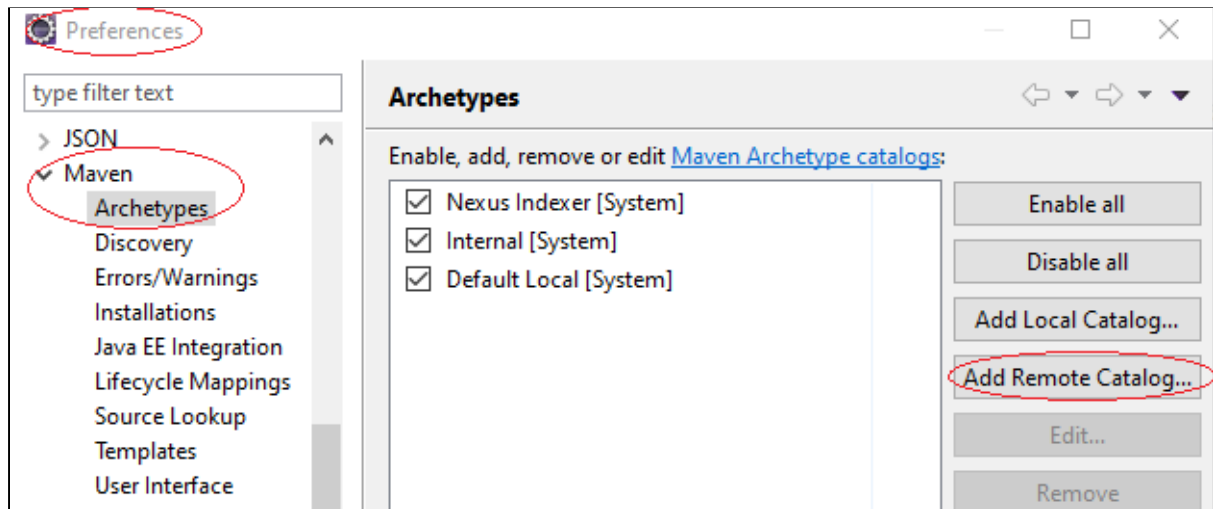
9. Annoter l'annotation @Programmer par l'annotation @Inherited et refaire l'exécution de la classe TestReflectionAnnotation. C'est quoi votre remarque?

Exercice :

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

V. Création d'un projet Maven

5. Ajoutez le catalogue Maven distant. [Il faut être connecté à Internet]
a. Windows -> Preferences -> Maven -> Archetypes->

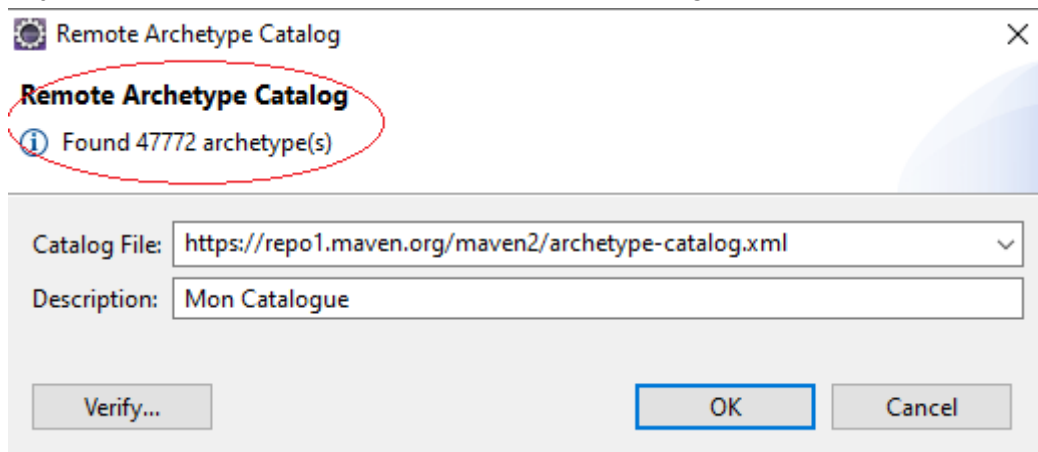


- b. Cliquez sur le bouton **Add Remote Catalog**.

Saisir le lien du fichier catalogue :

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

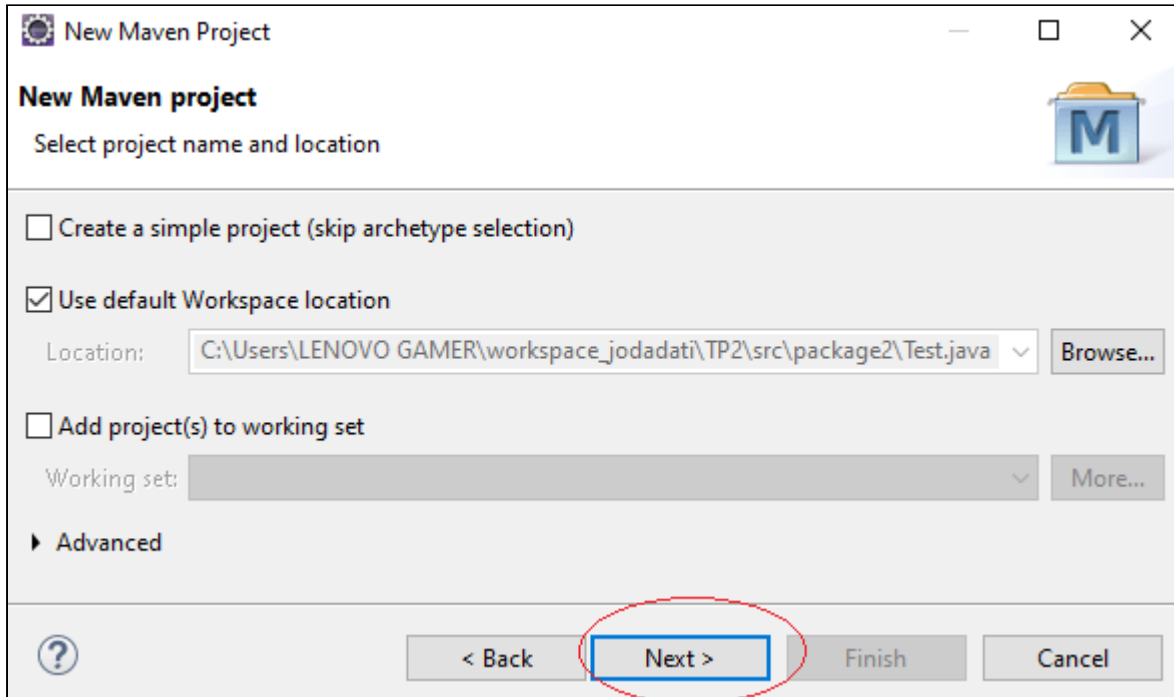
Ajouter une description de votre choix, "Mon Catalogue" à titre d'exemple



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

6. 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

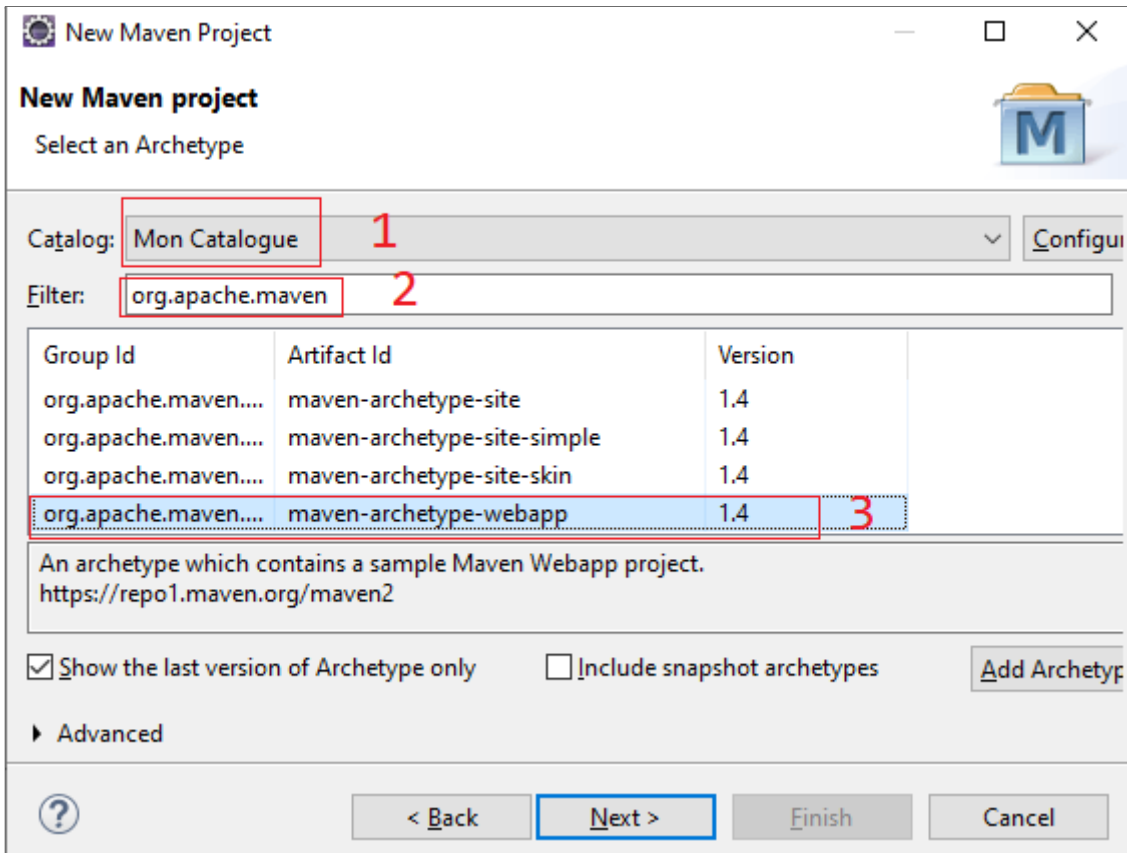
Location:

☐ 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

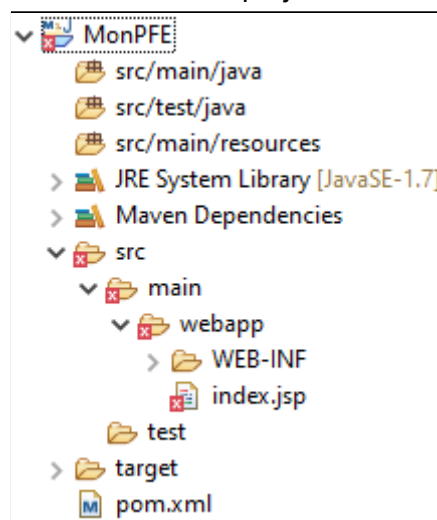
An archetype which contains a sample Maven Webapp project.
<https://repo1.maven.org/maven2>

☒ 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"

- 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 "ressources" 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:

```

v 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.

VI. Application des annotations : Lombok dependency

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

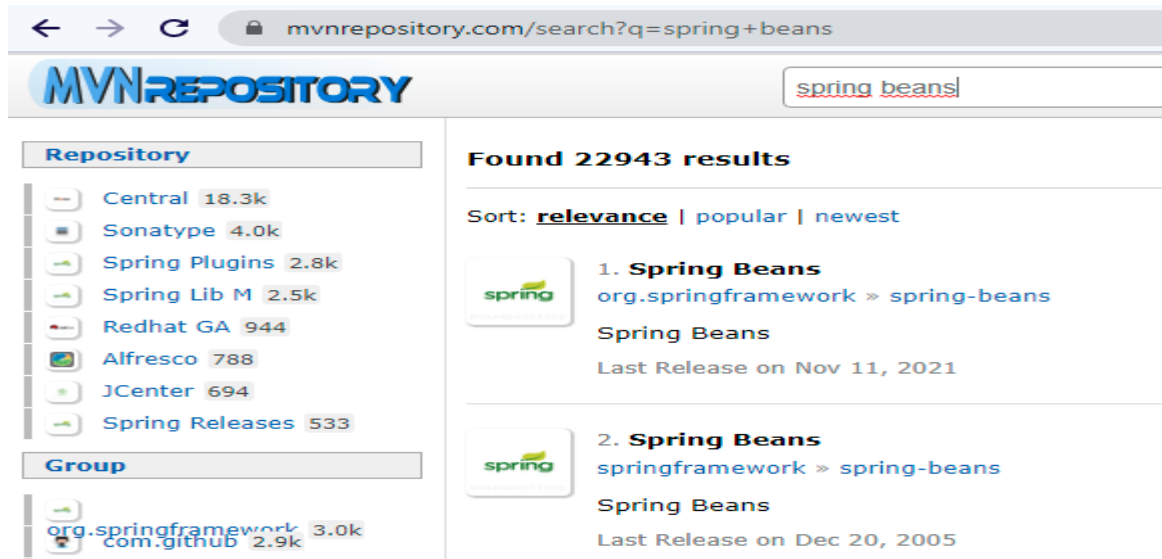
```
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <version>1.18.22</version>
</dependency>
```

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

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

VII. Ajout du Spring beans dependencies

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



← → ↺ mvnrepository.com/search?q=spring+beans

MVNREPOSITORY

Repository

- Central 18.3k
- Sonatype 4.0k
- Spring Plugins 2.8k
- Spring Lib M 2.5k
- Redhat GA 944
- Alfresco 788
- JCenter 694
- Spring Releases 533

Group

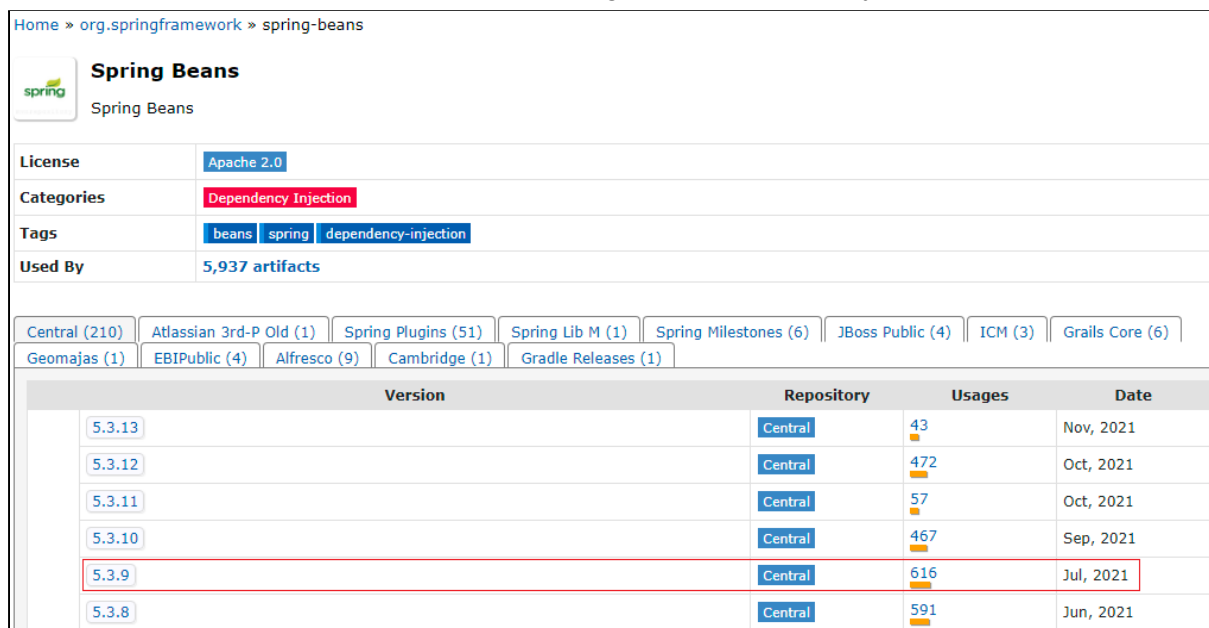
- org.springframework 3.0k
- com.github 2.9k

Found 22943 results

Sort: **relevance** | popular | newest

- Spring Beans**
org.springframework » spring-beans
Spring Beans
Last Release on Nov 11, 2021
- Spring Beans**
springframework » spring-beans
Spring Beans
Last Release on Dec 20, 2005

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)

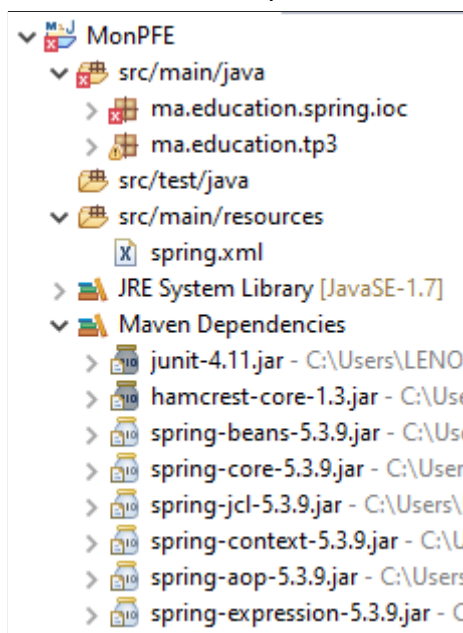
Version	Repository	Usages	Date
5.3.13	Central	43	Nov, 2021
5.3.12	Central	472	Oct, 2021
5.3.11	Central	57	Oct, 2021
5.3.10	Central	467	Sep, 2021
5.3.9	Central	616	Jul, 2021
5.3.8	Central	591	Jun, 2021

2. 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>
</dependencies>
```

```
<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



VIII. 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 TestSpringIocApplicationContext 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

instanciation 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`.

## IX. 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`

## X. 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`.

Dossier des travaux pratiques. Module 1 : Java de base .Années scolaire 2021/2022. Niveau : Licence FST Settat

Professeur : M. Boulchahoub Hassan [hboulchahoub@gmail.com](mailto:hboulchahoub@gmail.com)

Mise à jour 18 Nov 2021



- *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.*

## XI. 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.*