# Laborator 1: Dependency Injection, Inversion of Control

## DEPENDENCY INJECTION

**Dependency injection:** modul în care un obiect primește referințe către alte obiecte: o dependență (obiect helper) este inserată în obiectul unde este necesară. Obiectul în care este inserată nu este responsabil cu crearea dependenței.

Exemplu de dependență: o instanță a unui conexiuni la o baza de date.

O dependenta poate fi inserata (***injected***) prin trimiterea ca parametru al unui **constructor** (varianta preferata) sau prin intermediul unei metode de tip **setter**, ca alternativa la hardcodarea unei atribuiri, sau prin folosirea de adnotari la atributele clasei.

Tipuri de DI:
- **Constructor DI**
- **Setter DI**
- **Property DI**

Cum se poate configura:
- **XML configuration/Annotation DI**

**Dependency injection** poate fi utilizată cu clase sau cu interfețe. Este preferată utilizarea cu interfețe:
- se poate decide la runtime ce implementare a dependenței va fi inserata (*injected*).
- respecta principiul „Interface Segregation Principle" (***I*** din acronimul ***SOLID***)
- se pot executa ușor ***unit tests*** (se pot folosi mock objects)**.**

## INVERSION OF CONTROL

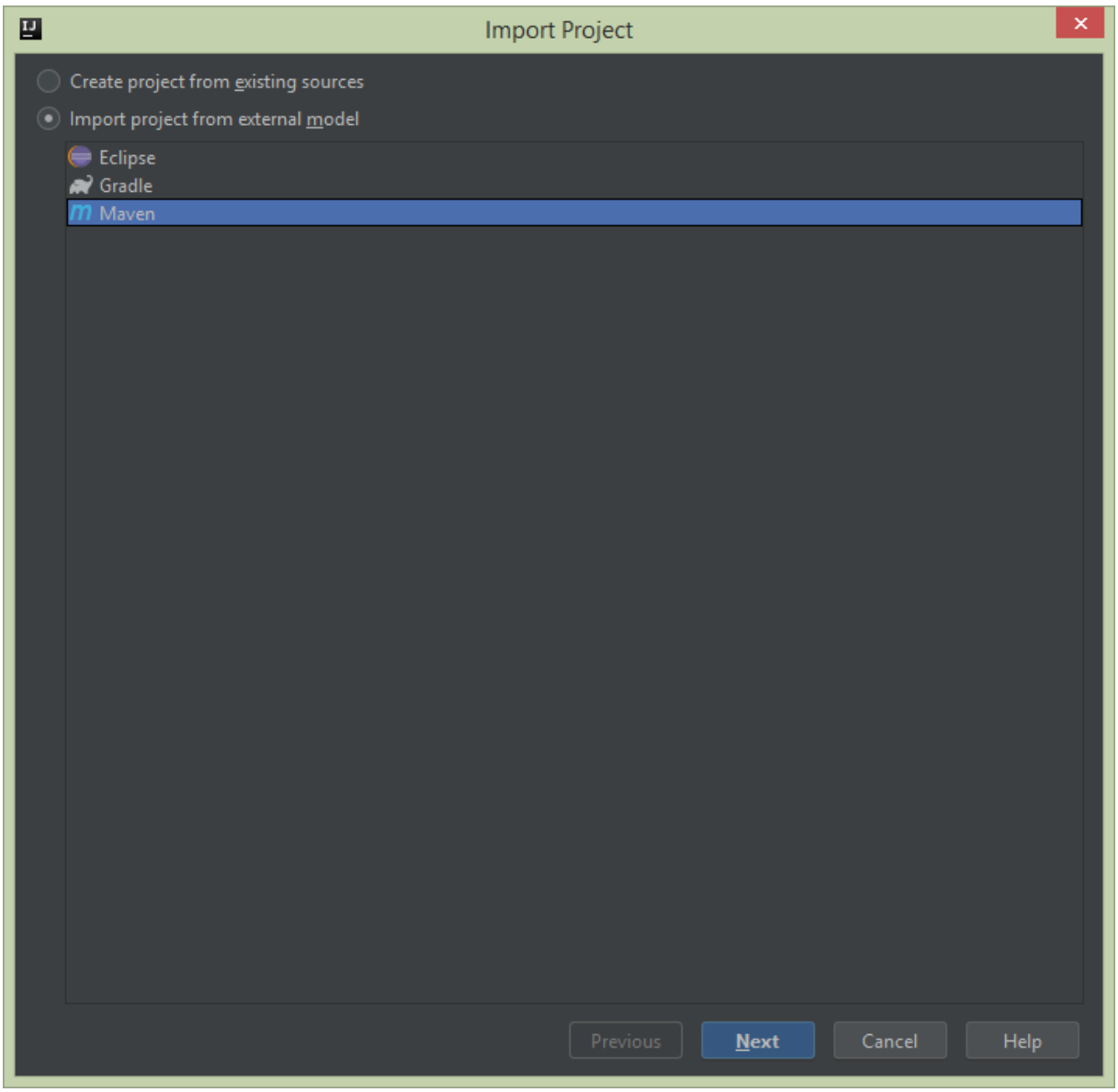
**IoC Inversion of Control** este mecanismul prin care întreg ciclul de viața al unui obiect este controlat de un *IoC Container*, începând de la crearea obiectelor până la distrugere, incluzând invocarea anumitor metode ale obiectelor (**callbacks**). Programatorul poate crea și manageria unele obiecte, unele însă vor fi furnizate de *IoC Container* la runtime.

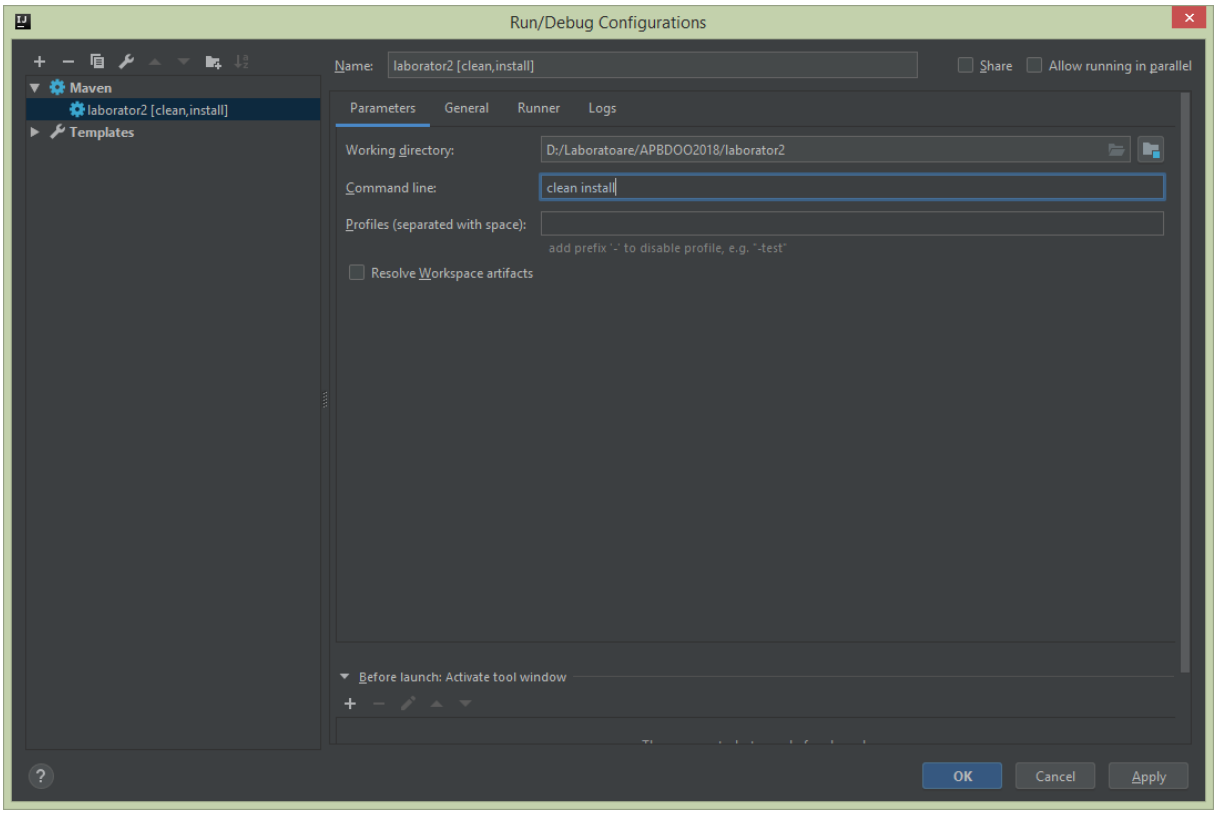
In Spring obiectele sunt controlate de **org.springframework.context.ApplicationContext** și se numesc **beans.**

Exemple de implementări ale **org.springframework.context.ApplicationContext**

- ClassPathXmlApplicationContext
- AnnotationConfigApplicationContext
- GenericApplicationContext
- GenericWebApplicationContext

1. Importați în IntelliJ proiectul lab1_start. Din meniu selectați File – New project from existing sources. Rulați mvn clean install, adăugați o configurație pentru a rula mvn clean install. Alternativ puteți crea un nou proiect maven (exercițiile 2 -7).






## MAVEN

**Maven** este un tool pentru optimizarea procesului de build al proiectelor Java, pentru gestionarea dependențelor și a plugin-urilor și downloadarea librariilor din unul sau mai multe *repository*-uri.

Maven asigură într-o manieră automată consistența versiunilor librariilor folosite între diferite etape ale dezvoltării aplicației și deploy, de asemenea asigură update-ul la cele mai recente versiuni ale modulelor folosite.

Alte exemple de tooluri utilizate pentru *dependency mangement* sunt: Gradle (pentru proiecte Java), npm (pentru pachete node), Composer (pentru proiecte php) etc.

Proiectele Maven sunt definite intr-un fisier **pom.xml.** Pom este abrevierea pentru „Project Object Model”.

2. Instalați **Maven**: https://maven.apache.org/download.cgi Dupa dezarhivarea arhivei apache-maven-3.6.0-bin.zip includeti in path:

C:\Program Files\apache-maven-3.6.0\bin.

Setati variabila JAVA_HOME:

C:\Program Files\Java\jdk1.8.0_11

Testati versiunea cu comanda (command prompt windows)

```
>> mvn -v
```

3. Structura unui proiect maven: creați directorul în care veti salva proiectul și un director în care veți retine fisierele sursa Java.

```
>> mkdir laborator1
>> mkdir ”src/main/java”
>> mkdir ”src/main/resources”
>> mkdir ”src/test/java”
>> mkdir ”src/test/resources”
>> mkdir ”target”
```

**POM** Proiectele Maven sunt definite intr-un fisier **pom.xml.** Pom este abrevierea pentru „Project Object Model”. Configuratia minimala a unui proiect "org.apbdoo.app:rest-service:1" include:

*Groupul proiectului*:
<groupId>com.apbdoo</ groupId>
*Id-ul proiectului*:
<artifactId>laborator1</artifactId>
*Versiunea*
<version>1</version> si
<modelVersion>**4.0.0** </modelVersion>.

Orice configuratie definită într-un pom moșteneste, direct sau indirect, configuratia pom-ului default (Super pom). Spre exemplu tipul de packing, jar <packaging>jar</packaging> este valoarea default mostenita de la Super pom.
Orice fisier pom poate avea un **parent**. Pom-urile parent vor avea <packaging>pom</packaging> si o sectiune <modules>.

Sectiunea <dependencies> include proiectele de care depinde proiectul, pentru fiecare dintre acestea specificandu-se grupul, id-ul, versiune si scope = compile | provided | test. Daca se specifica <scope>test</scope> pentru o librarie, aceasta nu va fi utilizata la runtime.

Sectiunea <build>…</build> include informatii despre structura proiectului, resurse si plugin-uri.

4. Creați următorul fișier pom.xml in directorul `laborator1`:

```
<?xml version="1.0" encoding="UTF-8"?>
<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>com.apbdoo</groupId>
    <artifactId>laborator1</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>
```

```
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>5.1.3.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.0.9.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

5. Executati in command prompt:

```
>> mvn compile
```

Verificati in directorul target generarea fisierului Hello.class si continutul repositoryului maven local **.m2**, spre exemplu C:\Users\username\.m2\repository.

**build lifecycle:** Procesul de build (build lifecycle) al unui proiect maven este impartit in etape. Exista trei lifecycle-uri predefinite: **default, clean, site**.
Lifecycle-ul **default** parcurge urmatoarele etape (faze):

| | |
|---|---|
| **validate** | verifica daca toate infomatiile necesare proiectului sunt disponibile |
| **compile** | compileaza codul sursa |
| **test** | testeaza codul sursa |
| **package** | arhiveaza codul, spre exemplu intr-un fisier .jar. |
| **verify** | realizeaza *integration tests* (se testeaza functionalitatile modulelor in mod unitar) |
| **install** | instaleaza proiectul in repository-ul local |
| **deploy** | instaleaza proiectul in repository-ul remote |

**phases** mvn poate executa o etapa (**phase**). Comanda mvn install va executa in ordine toate fazele care preced install si faza install.

**goals** mvn poate executa, de asemenea, prin intermediul plug-inurilor taskuri specifice (**goals**) care pot fi asociate unei anumite etape.

6. Executati in command prompt urmatoarele comenzi. Care este rolul lor? Care reprezinta taskuri specifice (goals)?

```
>> mvn clean
>> mvn jar:jar
>> mvn clean install
>> mvn clean install jar:jar
```

7. Configurați în fișierul de configurare **resources/applicationContext.xml** două *bean*-uri cu id-urile **myBooksSubscription** și **myMoviesSubscription**. Creați două teste JUnit în care să obțineți din context cele două instanțe și apelați metodele getDescription() și getPrice ale acestora.

8. Creați în pachetul com.apbdoo.lab1 o nouă implementare a interfeței Subscription. Ce metode vor fi implementate? Configurați un bean de tipul SportSubscription și testați

```
public class SportSubscription implements Subscription{
    …
}
```

9. Creați un nou fișier de configurare **applicationContextDI.xml** cu următorul conținut:

```
<?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:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-
context.xsd">

       <bean id="myMoviesSubscription"
             class="com.apbdoo.lab1.MoviesSubscription">
       </bean>

       <bean id="myBooksSubscription"
               class="com.apbdoo.lab1.BooksSubscription">
       </bean>

       <bean id="mySportSubscription"
               class="com.apbdoo.lab1.SportSubscription">
       </bean>

</beans>
```

10. Implementați interfața DiscountCalculator astfel încât metoda calculate să aplice prețului primit ca argument o reducere de 10%.

```
public class DiscountCalculatorImpl implements DiscountCalculator{
    …
}
```

11. Adăugați în clasa BooksSubscription un atribut de tipul DiscountCalculatorImpl discountCalculatorImpl. Utilizați metoda calculate în metodele getPrice()

```
DiscountCalculator discountCalculator;

public double getPrice() {
    return discountCalculator.calculate(100);
}
```

12. Adăugați în clasa BooksSubscription un constructor cu un argument de tipul DiscountCalculator și un constructor fără argumente. Puteți utiliza în IntelliJ-Code Generate-Constructor.

```
public BooksSubscription(DiscountCalculator discountCalculator) {
        this.discountCalculator = discountCalculator;
}
```

13. Configurați în fișierul **applicationContextDI.xml** argumentul pentru constructorul clasei BooksSubscription care va specifica implementarea interfeței DiscountCalculator care va fi injectat în clasa BooksSubscription.

```
<bean id="myMoviesSubscription"
      class="com.apbdoo.lab1.MoviesSubscription">
</bean>

<bean id="myBooksSubscription"
      class="com.apbdoo.lab1.BooksSubscription">
<constructor-arg name="discountCalculator" ref="myDiscountCalculator" />
</bean>
```

14. Creați clasa TestDI cu o metodă care să testeze obținerea din context a unei instanțe de tipul BooksSubscription.

```
@Test
public void contructorDI(){
    ClassPathXmlApplicationContext context =
  new ClassPathXmlApplicationContext("applicationContextDI.xml");
```

```
        Subscription theSubscription =
context.getBean("myBooksSubscription", Subscription.class);

        System.out.println(theSubscription.getPrice() + " " +
theSubscription.getDescription());

        context.close();
}
```

15. Adăugați în clasa MoviesSubscription un atribut de tipul DiscountCalculatorImpl discountCalculatorImpl. Utilizați metoda calculate în metodele getPrice()
16. Adaugați o metodă de tip setter pentru atributul discountCalculatorImpl. Puteți utiliza în IntelliJ-Code Generate-Setter.
17. Configurați bean-ul cu id-ul MoviesSubscription pentru a se injecta o implementare a interfeței DiscountCalculator. Scrieți un test care să obțină din context un obiect de tipul MoviesSubscription și să apeleze metodele getDescription și getPrice.

```
<bean id="myMoviesSubscription"
                    class="com.apbdoo.lab1.MoviesSubscription">
          <property name="discountCalculator" ref="myDiscountCalculator"/>
 </bean>
```

18. Adăugați fișierul **resources/application.properties** cu următorul conținut:

```
discount.percent=0.2
```

În fișierul applicationContextDI.xml adăugați:

```
<context:property-placeholder location =
"classpath:application.properties"/>
```

19. Adăugați în clasa DiscountCalculatorImpl atributul double percent. Valoarea acestuia va fi injectată printr-o metodă de tip setter din fișierul `application.properties`.

```
<bean id="myDiscountCalculator"
              class="com.apbdoo.lab1.DiscountCalculatorImpl">
              <property name="percent"
value="${discount.percent}"/>
</bean>
```

20. Configurați scope pentru bean-ul mySportSubscription și metodele care se vor apela la inițializare/destroy în fișierul **resources/applicationContext.xml**. Adaugați un test care obțină din context două bean-uri mySportSubscription.

```
<bean id="mySportSubscription"
                class="com.apbdoo.lab1.SportSubscription"
                scope = "prototype"
                init-method = "initSportSubscription"
                >
</bean>
```

```
public class LifeCycleTest {
    @Test
    public void testLifeCycle(){


        ClassPathXmlApplicationContext context =
                new
ClassPathXmlApplicationContext("applicationContext.xml");

        SportSubscription theSubscription =
context.getBean("mySportSubscription", SportSubscription.class);
        System.out.println(theSubscription.getPrice() + " " +
theSubscription.getDescription());


        SportSubscription theSubscription2 =
context.getBean("mySportSubscription", SportSubscription.class);
        System.out.println(theSubscription2.getPrice() + " " +
theSubscription2.getDescription());


        assertNotEquals(theSubscription, theSubscription2);
        context.close();

    }
}
```

21. Adăugați o metodă destroy. Se va apela pentru bean-uri cu scoupe = prototype?

```
<bean id="mySportSubscriptionSingleton"
                class="com.apbdoo.lab1.SportSubscription"
                scope = "singleton"
                init-method = "initSportSubscription"
                destroy-method = "destroySportSubscription">
      </bean>
```