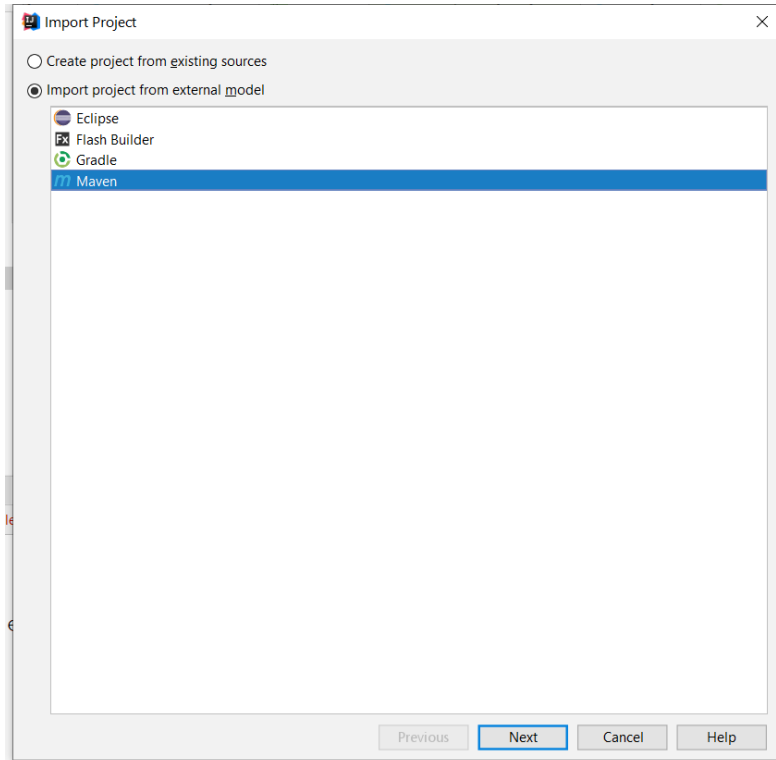


Practice 1: DI, IoC, Java beans

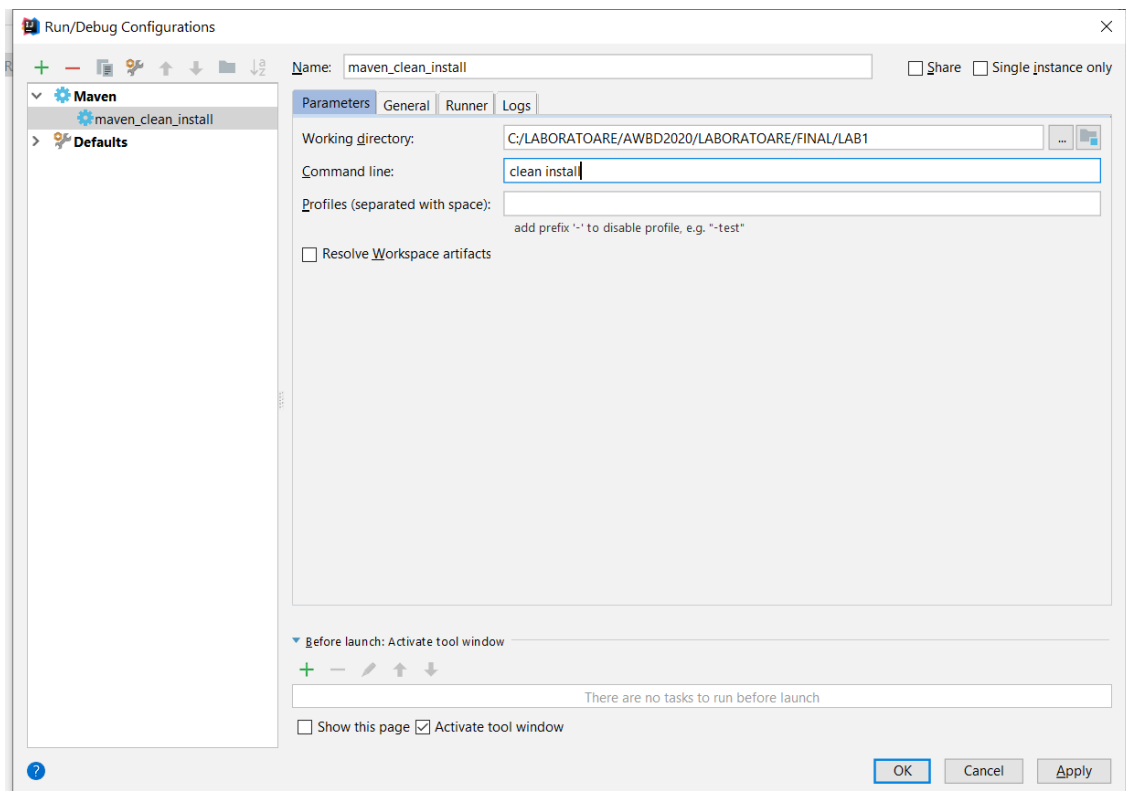
1.

Open the project lab1 in IntelliJ IDE: File -> New Project from Existing Sources. Import as Maven project:



2.

Set up Maven run configuration:



3.

Setup in resources/applicationContext.xml two beans with ids myBooksSubscription and myMoviesSubscription. Create two JUnit test to get the beans from the context and run methods getDescription() and getPrice().

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

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

```
@Test
public void testXmlContext() {

    // load the spring configuration file
    ClassPathXmlApplicationContext context =
        new
        ClassPathXmlApplicationContext("applicationContext.xml");

    // retrieve bean from spring container
    Subscription theSubscription = context.getBean("mySubscription",
        Subscription.class);

    // call methods on the bean
    System.out.println(theSubscription.getPrice() + " " +
        theSubscription.getDescription());

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

    // call methods on the bean
    System.out.println(theBooksSubscription.getPrice() + " " +
        theBooksSubscription.getDescription());

    Subscription theMoviesSubscription =
        context.getBean("myMoviesSubscription", Subscription.class);

    // call methods on the bean
    System.out.println(theMoviesSubscription.getPrice() + " " +
        theMoviesSubscription.getDescription());

    // close the context
    context.close();
}
```

Dependency injection [1]

Inversion of Control (IoC) objects do not create other objects on which they rely to do their work. Instead, they get the objects that they need from an outside source (for example, an xml configuration file).

Info

Spring container "injects" objects into other objects or "dependencies", a dependency is any object that another object requires.

Examples: database connections, view resolvers, services etc.

DI moves the responsibility of managing components onto the container. (Inversion of Control)

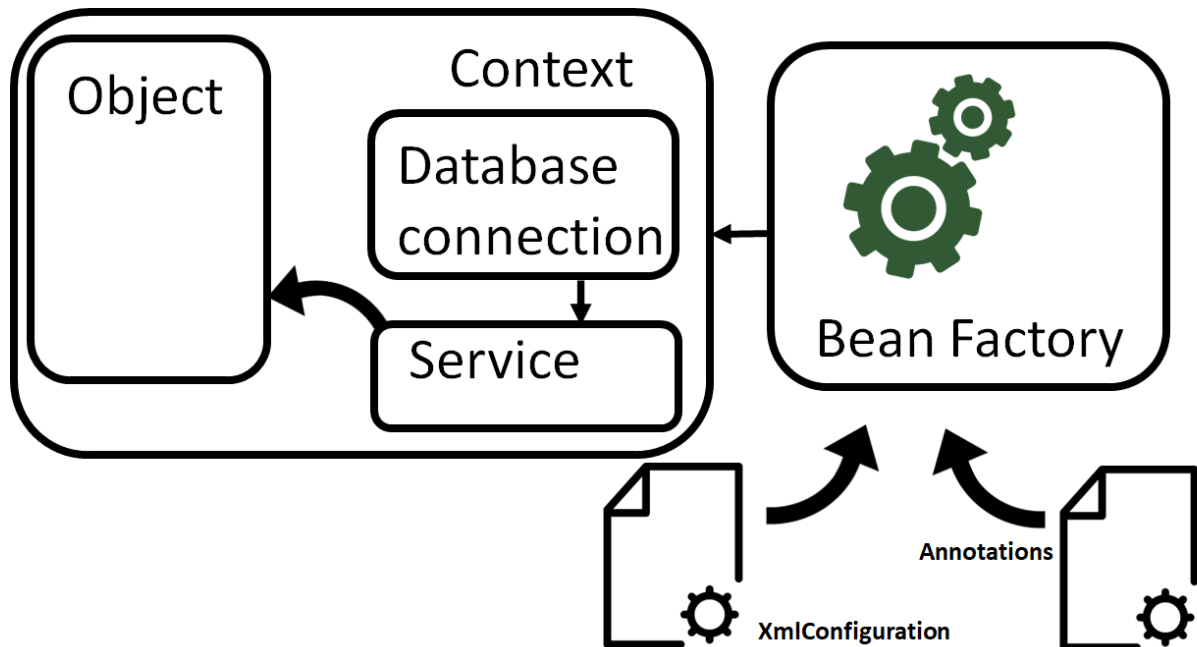
Info

Types of DI:

- setter DI
- constructor DI (implicit)
- property DI

DI works with *classes* or with *interfaces*. Advantages of using interfaces:

- The implementation is decided at runtime, we may use different implementations.
- Easy to test with mock objects.
- Loose coupling of components



ApplicationContext types [2][3]:

ClassPathXmlApplicationContext

Load an XML configuration file from the classpath.

FileSystemXMLApplicationContext

Load an XML configuration file from the filesystem or from urls.

XmlWebApplicationContext

Xml configuration for web application (file web.xml)

AnnotationConfigApplicationContext

Loads classes annotated with `@Configuration` or `@Component`

AnnotationConfigWebApplicationContext

Web-based variant of **AnnotationConfigApplicationContext**

4.

Implement interface DiscountCalculator

```
public class DiscountCalculatorImpl implements DiscountCalculator{

    public double calculate(int price) {
        return price * 0.9;
    }
}
```

5.

Add in BooksSubscription class an attribute DiscountCalculatorImpl discountCalculatorImpl. Use calculate method in getPrice method.

```
DiscountCalculator discountCalculator;

public double getPrice() {
    return discountCalculator.calculate(450);
}
```

6.

Add in BooksSubscription a constructor whit an argument of type DiscountCalculator and a no arg construtor. We may use IntelliJ-Code Generate-Constructor.

```
public BooksSubscription() {
}

public BooksSubscription(DiscountCalculator discountCalculator) {

    this.discountCalculator = discountCalculator;
}
```

7.

Create a new file ApplicationContextDI.xml , in resources. In the next steps we will configure the bean myDiscountCalculator.

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

</beans>
```

8.

Create file resources/application.properties:

```
discount.percent=0.2
```

9.

Add in ApplicationContextDI:

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

10.

Add an attribute double percent in DiscountCalculatorImpl and a setter method for this attribute.

```
double percent;

public void setPercent(double percent) {
    this.percent = percent;
}
```

11.

Configure myDiscountCalculator bean in ApplicationContextDI.xml

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

12.

Add a test method constructorDI and fix testXmlContext() test.

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

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

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

    context.close();
}
```

13.

Configure bean myMovieSubscription, use setter injection:

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

14.

Add setterDI() test and fix testXmlContext() test:

```

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

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

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

    context.close();
}

```

15.

Create file resources/applicationContextCS

```

<?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">

    <context:property-placeholder
        location="classpath:application.properties" />
    <context:component-scan base-package="com.awbd.lab1b" />

</beans>

```

16.

Create a new package com.awbd.lab1b and copy classes from com.awbd.lab1. Annotate class com.awbd.lab1b.SportSubscription with @Component. Annotate with component BooksSubscription and MoviesSubscription.

```

@Component("mySportSubscription")
public class SportSubscription implements Subscription

```

17.

Annotate percent property in DiscountCalculatorImpl

```

@Value("${discount.percent}")
double percent;

@Value("0.2")
double percent;

```

18.

Add attribute `discountCalculator` in `SportSubscription` and annotate it with `@Autowired`.
Add annotation `@Autowired` to `setDiscountCalculator` method in `MoviesSubscription`
Annotate `@Autowired` the constructor `BooksSubscription`.

```
@Autowired
DiscountCalculator discountCalculator;

public double getPrice() {
    return discountCalculator.calculate(1000);
}
```

```
@Autowired
public void setDiscountCalculator(DiscountCalculator discountCalculator)
{
    this.discountCalculator = discountCalculator;
}
```

Info

@Component (value=componentName) [4] is used with class-path scanning pentru înregistrarea bean-urilor to register beans in the context. If value attribute is not present the default bean id is the name of the class with lowercase letters.

Examples of components: **@Service**, **@Repository**, **@Controller**

@Primary [6] specifies the default bean to be injected as a dependency

@Qualifier [5] specifies the id of the bean to be injected.

19.

Annotate @Component DiscountCalculatorImpl. Add test class com.awbd.lab1b.ContextLoadTest:

```

package com.awbd.lab1b;
import com.awbd.lab1b.Subscription;
import org.junit.Test;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class ContextLoadTest {
    @Test
    public void constructorDI() {
        ClassPathXmlApplicationContext context =
            new
            ClassPathXmlApplicationContext("applicationContextCS.xml");

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

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

        context.close();
    }
    @Test
    public void setterDI() {
        ClassPathXmlApplicationContext context =
            new
            ClassPathXmlApplicationContext("applicationContextCS.xml");

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

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

        context.close();
    }
}

```

20.

Create a new implementation for DiscountCalculator interface. Rerun tests.

```

package com.awbd.lab1b;

import org.springframework.stereotype.Component;

@Component
public class FixDiscountCalculator implements DiscountCalculator{

    public double calculate(int price) {
        return 0.85 * price;
    }
}

```


21.

Add @Primary annotation in FixDiscountCalculator:

```
package com.awbd.lab1b;

import org.springframework.stereotype.Component;

@Component
@Primary
public class FixDiscountCalculator implements DiscountCalculator{

    public double calculate(int price) {
        return 0.85 * price;
    }
}
```

22.

Add Qualifier annotation to setter method and constructor.

```
@Autowired
@Qualifier("discountCalculatorImpl")
public void setDiscountCalculator(DiscountCalculator discountCalculator)
{
    this.discountCalculator = discountCalculator;
}
```

```
@Autowired
public BooksSubscription(@Qualifier("discountCalculatorImpl")
DiscountCalculator discountCalculator) {

    this.discountCalculator = discountCalculator;
}
```

23.

Add a configuration class:

```

package com.awbd.lab1b;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;

class ExternalCalculator implements DiscountCalculator{
    public double calculate(int price) {
        return 0.65 * price;
    }
}

@Configuration
@ComponentScan("com.awbd.lab1b")
@PropertySource("classpath:application.properties")
public class SubscriptionConfig {

    @Bean
    public DiscountCalculator externalCalculator(){
        return new ExternalCalculator();
    }

}

```

24.

Change @Qualifier annotation for setter method in MoviesSubscription:

```

@Autowired
@Qualifier("externalCalculator")
public void setDiscountCalculator(DiscountCalculator discountCalculator)
{
    this.discountCalculator = discountCalculator;
}

```

```

package com.awbd.lab1b;
import org.junit.Test;
import
org.springframework.context.annotation.AnnotationConfigApplicationContex
t;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

public class ContextLoadTestAnnotation {

    @Test
    public void constructorDI() {
        AnnotationConfigApplicationContext context =
            new
AnnotationConfigApplicationContext(SubscriptionConfig.class);

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

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

        context.close();
    }

    @Test
    public void setterDI() {
        AnnotationConfigApplicationContext context =
            new
AnnotationConfigApplicationContext(SubscriptionConfig.class);
        Subscription theSubscription =
context.getBean("myMoviesSubscription", Subscription.class);
        System.out.println(theSubscription.getPrice() + " " +
theSubscription.getDescription());

        context.close();
    }

    @Test
    public void fieldDITest() {
        AnnotationConfigApplicationContext context =
            new
AnnotationConfigApplicationContext("com.awbd.lab1b");
        Subscription theSubscription =
context.getBean("mySportSubscription", Subscription.class);
        System.out.println(theSubscription.getPrice() + " "
            + theSubscription.getDescription());
        context.close();
    }
}

```

B

- [1] <https://www.baeldung.com/spring-dependency-injection>
- [2] <https://www.baeldung.com/spring-application-context>
- [3] <https://docs.spring.io/spring-framework/docs/5.3.x/reference/html/core.html#beans>
- [4] <https://www.baeldung.com/spring-component-repository-service>
- [5] <https://www.baeldung.com/spring-qualifier-annotation>
- [6] <https://www.baeldung.com/spring-primary>
- [7] <https://www.baeldung.com/spring-component-scanning>
- [8] <https://www.baeldung.com/circular-dependencies-in-spring>