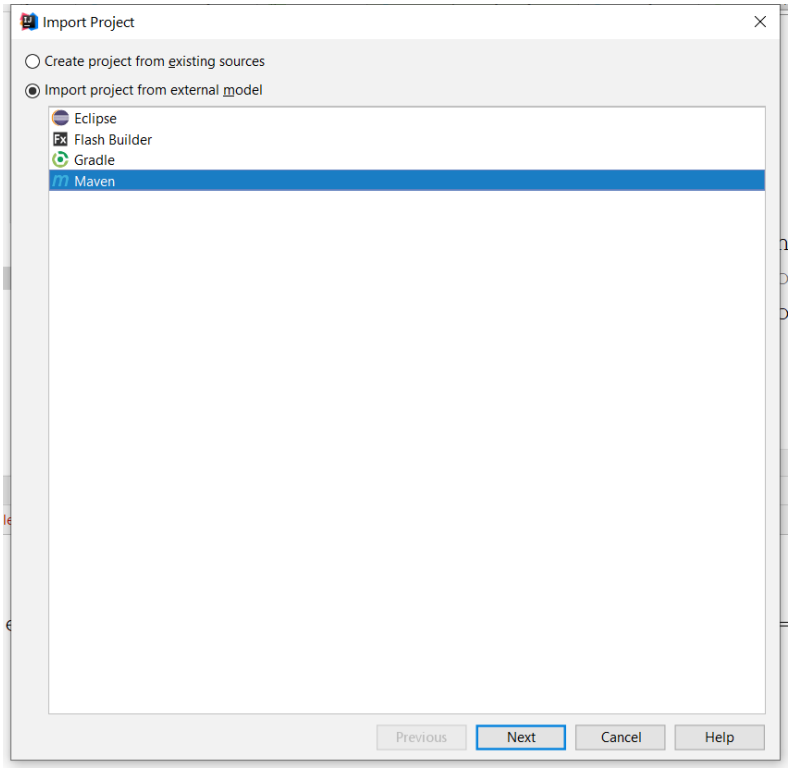
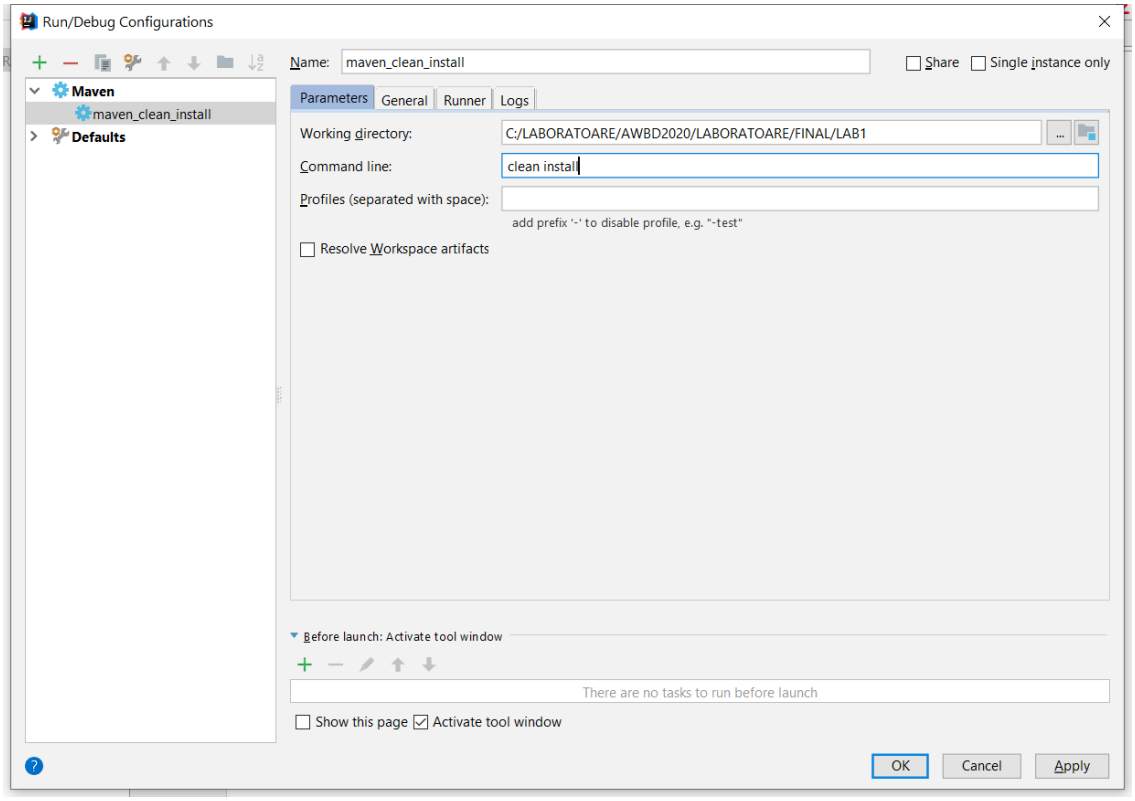
Practice 1: DI, IoC, Java beans

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 tests 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 mySportSubscription =
context.getBean("mySportSubscription", Subscription.class);

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

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

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

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

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

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

Info

Dependency injection and Inversion of Control [1]

Inversion of Control (IoC) is a principle in software engineering, most often used in the context of object-oriented programming, which transfers the control of objects to an outside source (container or framework). Objects do not create other objects on which they depend or rely to do their work. Instead, they get the objects that they need from the outside source (for instance Spring Container). The source is responsible for creating objects with specific properties specified in a configuration file or by annotations.

Info

Dependency Injection is a pattern used to implements IoC. Dependency Injection is a fundamental aspect of the Spring framework through which the Spring container “injects” injects objects into other objects or “dependencies”. This allows loose coupling of components and moves the responsibility of managing components onto the container. 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.

The advantages of IoC and DI are:

- Loose coupling of components.
- Use of interchangeable implementations.
- Ease in creating mocks of dependencies and creating test.

In traditional object-oriented programming when we create an object we must instantiate all object that are encapsulated by it. For example, if we need an instance of type Provider we must first create an instance of type Service.

```
public class Provider{
        private Service service;

        public Provider(){
                service = new ServiceImplementation1();
        }
}
```

By using DI, we can define a type *Provider*, without specifying any implementations for the type Service.

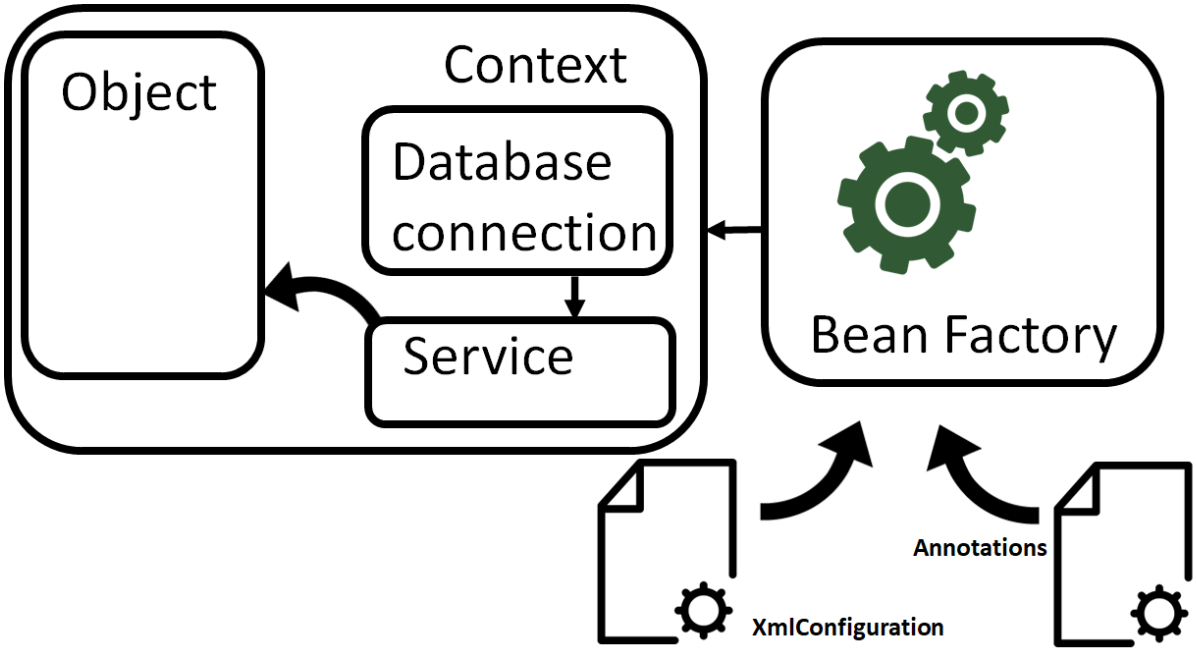
```
public class Provider{
        private Service service;

        public Provider(Service service){
                this.service = service;
        }
}
```

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

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

Types of DI:

- **constructor DI (implicit)**: the container will invoke a constructor with arguments, each argument represents a dependency. Constructor injection is used for mandatory dependencies.
- **setter DI:** The dependencies are provided by setter methods. Setter methods are called after the invocation of no-argument constructor. Setter based DI is used for optional dependencies.
- **property DI**: Uses reflection and @Autowired annotation. It’s a simpler but costlier approach and has the drawback that by using it we may easily violate the “single responsibility principle”, adding multiple dependencies in a single object.



In the Spring framework, the interface ApplicationContext represents the IoC container. The objects instantiated, configurated and assembled by the ApplicationContext are known as **beans**. **ApplicationContext** manages beans' life cycles and has the following implementations [2][3], depending on the form configuration metadata is read by the container:

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 *DiscountCalculator* discountCalculator. 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-argument constructor. 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;
}

public double calculate(int price) {
    return price * (1- percent);
}
```

11. Configure myDiscoutCalculator 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 testXmlContex() test.

```
@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();
}
```

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 testXmlContex() 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

```
<?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
https://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 all classes (BooksSubscription, MoviesSubscription, SportSubscription) with @Component.

```
@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 *setDiscoutCalculator* method in *MoviesSubsciption*
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 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 type to be injected as a dependency in case there are multiple implementations for the dependency.

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

19. Annotate @Component the implementation *DiscountCalculatorImpl*. Add test class com.awbd.lab1b.ContextLoadTest:

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

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

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

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

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

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

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

        context.close();
    }


    @Test
    public void propertyDITest(){
        ClassPathXmlApplicationContext context =
                new
ClassPathXmlApplicationContext("applicationContextCS.xml");
        Subscription mySportSubscription =
context.getBean("mySportSubscription", Subscription.class);
        System.out.println(mySportSubscription.getPrice() + " "
                + mySportSubscription.getDescription());
        context.close();
    }

}
```

20. Create a new implementation for DiscountCalculator interface. Rerun tests. Notice that only test for package lab1 pass.

```
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 anntoation to setter method and contructor.

```
@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;
}
```

25. Add test class `ContextLoadTestAnnotation`

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

public class ContextLoadTestAnnotation {

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

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

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

        context.close();
    }

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

        context.close();
    }

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