

Springboot: Dependency Injection

What is Dependency Injection

- Using Dependency Injection, we can make our class independent of its dependencies.
- It helps to remove the dependency on concrete implementation and inject the dependencies from external source.

Let's see the Problem first:

```
public class User {
    Order order = new Order();

    public User(){
        System.out.println("initializing user");
    }
}

public class Order {
    public Order(){
        System.out.println("initializing Order");
    }
}
```

Issues with above class structure:

- Both User and Order class are **Tightly coupled**.

- Suppose, Order object creation logic gets changed (lets say in future Object becomes an Interface and it has many concrete classes), then USER class has to be changed too.



- It breaks **Dependency Inversion Principle (DIP)**

- This principle says that DO NOT depend on concrete implementation, rather depends on abstraction.

Breaks Dependency Inversion Principle (DIP)

```
public class User {
    Order order = new OnlineOrder();

    public User(){
        System.out.println("initializing user");
    }
}
```

Follows Dependency Inversion Principle (DIP)

```
public class User {
    Order order;

    public User(Order orderObj) {
        this.order = orderObj;
    }
}
```

Now in Spring boot how to achieve Dependency Inversion Principle?

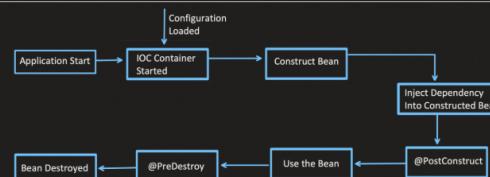
Through Dependency Injection

- Using Dependency Injection, we can make our class independent of its dependencies.
- It helps to remove the dependency on concrete implementation and inject the dependencies from external source.

```
@Component
public class User {
    @Autowired
    Order order;
}

@Component
public class Order {
}
```

@Autowired, first look for a bean of the required type.
-> If bean found, Spring will inject it.



Different ways of Injection and which one is better?

- Field Injection
- Setter Injection
- Constructor Injection

Field Injection

- Dependency is set into the fields of the class directly.
- Spring uses reflection, it iterates over the fields and resolve the dependency.

```
@Component
public class User {
    @Autowired
    Order order;

    public User(){
        System.out.println("User initialized");
    }
}
```

```
@Component
@Lazy
public class Order {
    public Order(){
        System.out.println("order initialized");
    }
}
```

[Report Abuse](#)

```

2024-04-13T21:33:30.398+05:30 INFO 20786 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080 (http)
2024-04-13T21:33:30.399+05:30 INFO 20786 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/] : StandardEngine [Tomcat/10.0.19]
2024-04-13T21:33:30.425+05:30 INFO 20786 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/] : Starting Servlet engine: [Apache Tomcat/10.0.19]
2024-04-13T21:33:30.425+05:30 INFO 20786 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 446 ms
User initialized
order initialized
2024-04-13T21:33:30.594+05:30 INFO 20786 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path ''
2024-04-13T21:33:30.595+05:30 INFO 20786 --- [           main] c.o.t.SpringbootApplication : Started SpringbootApplication in 0.811 seconds (process running for 0.983)

```

Advantage:

- Very simple and easy to use.

Disadvantage:

- Can not be used with immutable fields.

```

@Component
public class User {

    @Autowired
    public final Order order;

    public User() {
        System.out.println("User initialized");
    }
}

```

- Chances of NPE

```

@Component
public class User {

    @Autowired
    public Order order;

    public User() {
        System.out.println("User initialized");
    }

    public void process(){
        order.process();
    }
}

```

```

User userObj = new User();
userObj.process();

```

```

Exception in thread "main" java.lang.NullPointerException Create breakpoint :
at com.conceptandcoding.learningspringboot.User.process(User.java:18)

```

- During Unit Testing, setting MOCK dependency to this field becomes difficult.

```

@Component
public class User {

    @Autowired
    private Order order;

    public User() {
        System.out.println("User initialized");
    }

    public void process(){
        order.process();
    }
}

```

```

class UserTest {

    private Order orderMockObj;
    private User user;

    @BeforeEach
    public void setup(){
        this.orderMockObj = Mockito.mock(Order.class);
        this.user = new User(); ?? How to set this MOCK, we have to use reflection like
        // @InjectMock annotation internally uses
    }
}

```

```

class UserTest {

    @Mock
    private Order orderMockObj;

    @InjectMocks
    private User user;

    @BeforeEach
    public void setup(){
        MockitoAnnotations.initMocks(this);
    }
}

```

Setter Injection

- Dependency is set into the fields using the setter method.
- We have to annotate the method using @Autowired

```

@Component
public class User {

    public Order order;

    public User() {
        System.out.println("User initialized");
    }

    @Autowired
    public void setOrderDependency(Order order){
        this.order = order;
    }
}

```

```

@Component
@Lazy
public class Order {

    public Order(){
        System.out.println("order initialized");
    }
}

```

Advantage:

- Dependency can be changed any time after the object creation (as object can not be marked as final).
- Ease of testing, as we can pass mock object in the dependency easily.

Disadvantage:

- Field Can not be marked as final. (We can not make it immutable).

```

@Component
public class User {

    public final Order order;

    public User() {

```

```

        System.out.println("User initialized");
    }

    @Autowired
    public void setOrderDependency(Order order){
        this.order = order;
    }
}

```

- Difficult to read and maintained, as per standard, object should be initialized during object creation, so this might create code readability issue.

Constructor Injection

- Dependency get resolved at the time of initialization of the Object itself.
- Its recommended to use

```

@Component
public class User {
    Order order;

    @Autowired
    public User(Order order) {
        this.order = order;
        System.out.println("User initialized");
    }
}

```

```

@Component
@Lazy
public class Order {

    public Order(){
        System.out.println("order initialized");
    }
}

```

```

2024-04-13T21:08:45.923+05:30 INFO 19992 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080 (http)
2024-04-13T21:08:45.929+05:30 INFO 19992 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2024-04-13T21:08:45.929+05:30 INFO 19992 --- [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.19]
2024-04-13T21:08:45.955+05:30 INFO 19992 --- [main] o.a.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2024-04-13T21:08:45.955+05:30 INFO 19992 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 422 ms
User initialized
order initialized
2024-04-13T21:08:46.181+05:30 INFO 19992 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path ''
2024-04-13T21:08:46.185+05:30 INFO 19992 --- [main] c.c.l.SpringApplication : Started SpringbootApplication in 0.759 seconds (process running for 0.932)

```

When only 1 constructor is present, then using @Autowired on constructor is not mandatory. (from Spring version 4.3)

```

@Component
public class User {
    Order order;

    public User(Order order ) {
        this.order = order;
        System.out.println("User initialized");
    }
}

```

```

@Component
@Lazy
public class Order {

    public Order(){
        System.out.println("order initialized");
    }
}

```

```

2024-04-13T21:08:45.923+05:30 INFO 19992 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2024-04-13T21:08:45.929+05:30 INFO 19992 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Root WebApplicationContext: initialization completed in 422 ms
User initialized
order initialized
2024-04-13T21:08:46.181+05:30 INFO 19992 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Tomcat started on port 8080 (http) with context path ''
2024-04-13T21:08:46.185+05:30 INFO 19992 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Started SpringbootApplication in 0.759 seconds (process running for 0.932)

```

When more than 1 constructor is present, then using @Autowired on constructor is mandatory.

```

@Component
public class User {
    Order order;
    Invoice invoice;

    public User(Order order ) {
        this.order = order;
        System.out.println("User initialized with only Order");
    }

    public User(Invoice invoice ) {
        this.invoice = invoice;
        System.out.println("User initialized with only Invoice");
    }
}

```

```

@Component
@Lazy
public class Invoice {

    public Invoice() { System.out.println("invoice initialized"); }

}

```

Caused by: org.springframework.beans.BeanInstantiationException Create breakpoint : Failed to instantiate [com.conceptandcoding.learningspringboot.User]: No default constructor found

```

@Component
public class User {
    Order order;
    Invoice invoice;

    public User(Order order ) {
        this.order = order;
        System.out.println("User initialized with only Order");
    }

    @Autowired
    public User(Invoice invoice ) {
        this.invoice = invoice;
        System.out.println("User initialized with only Invoice");
    }
}

2024-04-13T21:16:01.024+05:30 INFO 20242 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2024-04-13T21:16:01.044+05:30 INFO 20242 --- [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.19]
2024-04-13T21:16:01.044+05:30 INFO 20242 --- [main] o.a.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2024-04-13T21:16:01.044+05:30 INFO 20242 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 430 ms
Invoice initialized
User initialized with only Invoice
2024-04-13T21:16:01.027+05:30 INFO 20242 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path ''
2024-04-13T21:16:01.032+05:30 INFO 20242 --- [main] c.c.l.SpringApplication : Started SpringbootApplication in 0.776 seconds (process running for 0.94)

```

Why Constructor Injection is Recommended (Advantages):

1. All mandatory dependencies are created at the time of initialization itself. Makes 100% sure that our object is fully initialized with mandatory dependency
 - i. avoid NPE during runtime
 - ii. Unnecessary null checks can be avoided too.

2. We can create immutable object using Constructor injection.

```

@Component
public class User {
    public final Order order;

    @Autowired
    public User(Order order) {
        this.order = order;
        System.out.println("User initialized");
    }
}

```

```

@Component
public class User {

    @Autowired
    public final Order order;

    public User() {
        System.out.println("User initialized");
    }
}

```

3. Fail Fast: if there is any missing dependency, it will fail during compilation itself, rather than failing during run Time.

```
@Component
public class User {
    public Order order;
    public User() {
        System.out.println("User initialized");
    }
    @PostConstruct
    public void init(){
        System.out.println(order == null);
    }
}
```

```
@Component
public class Order {
    public Order(){
        System.out.println("order initialized");
    }
}
```

Using Constructor Injection, even if we missed @Autowired

```
@Component
public class User {
    public Order order;
    public User(Order order) {
        this.order = order;
        System.out.println("User initialized");
    }
    @PostConstruct
    public void init(){
        System.out.println(order == null);
    }
}
```

OR

(it will fail fast, if Order bean is missing)

```
@Component
public class User {
    public Order order;
    public User(Order order) {
        this.order = order;
        System.out.println("User initialized");
    }
    @PostConstruct
    public void init(){
        System.out.println(order == null);
    }
}
```

```
public class Order {
    public Order(){
        System.out.println("order initialized");
    }
}
```

```
*****
APPLICATION FAILED TO START
*****
Description:
Parameter 0 of constructor in com.conceptunderstanding.learninginjectionpoint.User required a bean of type "com.conceptunderstanding.learninginjectionpoint.Order" that could not be found.
```

4. Unit testing is easy.

```
@Component
public class User {
    private Order order;
    @Autowired
    public User(Order order) {
        this.order = order;
        System.out.println("User initialized");
    }
    public void process(){
        order.process();
    }
}
```

```
class UserTest {
    private Order orderMockObj;
    private User user;
    @BeforeEach
    public void setup(){
        this.orderMockObj = Mockito.mock(Order.class);
        this.user = new User(orderMockObj);
    }
}
```

Common Issues when dealing with Dependency Injection:

1. CIRCULAR DEPENDENCY



Solutions:

1. First and foremost, can we refactor the code and remove this cycle dependency:

For example, common code in which both are dependent, can be taken out to separate class. This way we can break the circular dependency.

2. Using @Lazy on @Autowired annotation .

Spring will create proxy bean instead of creating the bean instance immediately during application startup.

@Lazy on field injection

Let's first consider this

```

@Component
@Lazy
public class Order {
    public Order() {
        System.out.println("Order initialized");
    }
}

@Component
public class Invoice {
    @Autowired
    public Order order;

    public Invoice() {
        System.out.println("Invoice initialized");
    }
}

2024-04-14T18:21:55.967+05:30 INFO 48155 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2024-04-14T18:21:55.967+05:30 INFO 48155 --- [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.19]
2024-04-14T18:21:55.993+05:30 INFO 48155 --- [main] o.a.c.c.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2024-04-14T18:21:55.993+05:30 INFO 48155 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 438 ms
Invoice initialized
Order initialized
2024-04-14T18:21:56.141+05:30 INFO 48155 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path ''
2024-04-14T18:21:56.144+05:30 INFO 48155 --- [main] c.c.l.SpringbootApplication : Started SpringbootApplication in 0.778 seconds (process running for 0.967)

```

Now, let's see this:

```

@Component
@Lazy
public class Order {
    public Order() {
        System.out.println("Order initialized");
    }
}

@Component
public class Invoice {
    @Lazy
    @Autowired
    public Order order;

    public Invoice() {
        System.out.println("Invoice initialized");
    }
}

2024-04-14T18:24:03.478+05:30 INFO 48250 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080 (http)
2024-04-14T18:24:03.478+05:30 INFO 48250 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2024-04-14T18:24:03.482+05:30 INFO 48250 --- [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.19]
2024-04-14T18:24:03.507+05:30 INFO 48250 --- [main] o.a.c.c.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2024-04-14T18:24:03.507+05:30 INFO 48250 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 426 ms
Invoice initialized
Order initialized
2024-04-14T18:24:03.677+05:30 INFO 48250 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path ''
2024-04-14T18:24:03.683+05:30 INFO 48250 --- [main] c.c.l.SpringbootApplication : Started SpringbootApplication in 0.792 seconds (process running for 0.96)

```

Now, We can use this `@Lazy` to resolve the circular dependency

```

@Component
public class Order {
    @Autowired
    Invoice invoice;

    public Order() {
        System.out.println("order initialized");
    }
}

@Component
public class Invoice {
    @Lazy
    @Autowired
    public Order order;

    public Invoice() {
        System.out.println("Invoice initialized");
    }
}

2024-04-14T18:27:04.567+05:30 INFO 48425 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2024-04-14T18:27:04.568+05:30 INFO 48425 --- [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.19]
2024-04-14T18:27:04.591+05:30 INFO 48425 --- [main] o.a.c.c.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2024-04-14T18:27:04.592+05:30 INFO 48425 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 432 ms
Invoice initialized
Order initialized
2024-04-14T18:27:04.745+05:30 INFO 48425 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path ''
2024-04-14T18:27:04.750+05:30 INFO 48425 --- [main] c.c.l.SpringbootApplication : Started SpringbootApplication in 0.784 seconds (process running for 0.958)

```

3. Using `@PostConstruct`

```

@Component
public class Order {
    @Autowired
    Invoice invoice;

    public Order() {
        System.out.println("Order initialized");
    }

    @PostConstruct
    public void initialize(){
        invoice.setOrder(this);
    }
}

@Component
public class Invoice {
    public Order order;

    public Invoice() {
        System.out.println("Invoice initialized");
    }

    public void setOrder(Order order) {
        this.order = order;
    }
}

2024-04-14T18:27:04.507+05:30 INFO 48425 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2024-04-14T18:27:04.508+05:30 INFO 48425 --- [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.19]
2024-04-14T18:27:04.591+05:30 INFO 48425 --- [main] o.a.c.c.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2024-04-14T18:27:04.592+05:30 INFO 48425 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 432 ms
Invoice initialized
Order initialized
2024-04-14T18:27:04.745+05:30 INFO 48425 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path ''
2024-04-14T18:27:04.750+05:30 INFO 48425 --- [main] c.c.l.SpringbootApplication : Started SpringbootApplication in 0.784 seconds (process running for 0.958)

```

UNSATISFIED DEPENDENCY

Problem:
`[Component]` `public interface Order {`

APPLICATION FAILED TO START

UnsatisfiedDependencyException: Error creating bean with name 'user'

Solution:

1. @Primary annotation

```

@Component
public class User {
    @Autowired
    Order order;

    public User() {
        System.out.println("User initialized");
    }
}
```

```

@Primary
@Component
public class OnlineOrder implements Order {
    public OnlineOrder() {
        System.out.println("Online order initialized");
    }
}
```

```

@Component
public class OfflineOrder implements Order{
    public OfflineOrder() {
        System.out.println("Offline order initialized");
    }
}

```

2. @Qualifier annotation

```

@Component
public class User {
    @Qualifier("offlineOrderName")
    @Autowired
    Order order;

    public User() {
        System.out.println("User initialized");
    }
}
```

```

@Component
@Qualifier("onlineOrderName")
public class OnlineOrder implements Order {
    public OnlineOrder() {
        System.out.println("Online order initialized");
    }
}

```

```

@Component
@Qualifier("offlineOrderName")
public class OfflineOrder implements Order{
    public OfflineOrder() {
        System.out.println("Offline order initialized");
    }
}

```

2024-04-14T19:10:15.437+01:00 [INFO] 51489 --- [main] o.a.a.c.c.t.TomcatEmbeddedServletContainer : Starting service [Tomcat]
2024-04-14T19:10:15.437+01:00 [INFO] 51489 --- [main] o.a.a.c.c.t.TomcatEmbeddedServletContainer : Initializing Spring embedded WebApplicationContext
2024-04-14T19:10:15.437+01:00 [INFO] 51489 --- [main] o.a.a.c.c.t.TomcatEmbeddedServletContainer : Root WebApplicationContext: initialization completed in 417 ms
2024-04-14T19:10:15.437+01:00 [INFO] 51489 --- [main] o.a.a.c.c.t.TomcatEmbeddedServletContainer : Tomcat started on port 8080 (http) with context path ""
2024-04-14T19:10:15.437+01:00 [INFO] 51489 --- [main] o.a.a.c.c.t.TomcatEmbeddedServletContainer : Started SpringBootApplication in 0.763 seconds (process running for 0.932)