올인원
스프링 프레임워크

Chapter 05
의존 객체 자동 주입

목차

학습목표

- 의존 객체를 자동 주입하는 방법에 대해서 학습합니다.
- @Autowired 애너테이션을 이용한 의존 객체 자동 주입을 학습합니다.
- @Qualifier 애너테이션을 이용한 의존 객체 자동 주입을 학습합니다.
- @Resource 애너테이션을 이용한 의존 객체 자동 주입을 학습합니다.
- @Inject 애너테이션을 이용한 의존 객체 자동 주입을 학습합니다.

Section 01

예제 프로젝트 준비: 스마트폰 연락처

1. 스마트폰 연락처의 개요

■ 시나리오

① 연락처 3개의 샘플 데이터를 데이터베이스에 등록한다.

② 전체 연락처 정보를 출력한다.

③ 특정 인물에 대한 연락처를 출력한다.

■ 프로그램 흐름도

그림 5-1 스마트폰 연락처 프로그램의 흐름도

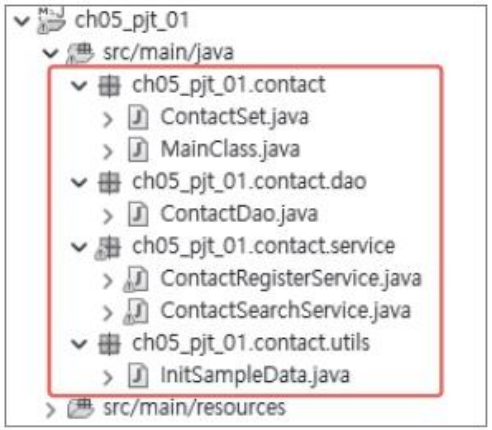
■ appCtx.xml 파일

- IoC 컨테이너 설정 파일
- IoC 컨테이너의 역할
  - ✓ Service 객체를 생성하고, Service가 생성될 때 필요한 DAO 객체를 생성해서 Service에 주입함

MainClass 클래스	
MainClass	main( ) 메서드가 명시되어 있는 클래스
Service 클래스	
ContactRegisterService	연락처 등록
ContactSearchService	연락처 조회
DAO 클래스	
ContactDao	DAO
ContactSet	연락처 1개 정보
Util 클래스	
InitSampleData	샘플 데이터 초기화
IoC 컨테이너	
appCtx.xml	객체 생성과 조립

그림 5-2 스마트폰 연락처 프로그램에서 사용될 클래스와 IoC 컨테이너 제작에 필요한 xml 파일

■ ch05_pjt_01 프로젝트에 패키지와 클래스 파일 생성



패키지	클래스
ch05_pjt_01.contact	ContactSet MainClass
ch05_pjt_01.contact.dao	ContactDao
ch05_pjt_01.contact.service	ContactRegisterService ContactSearchService
ch05_pjt_01.contact.utils	InitSampleData

그림 5-3 스마트폰 연락처 프로젝트의 패키지와 클래스

2. 스마트폰 연락처 만들기

ContactSet 클래스

- 연락처 정보를 담고 있음
  - ✓ name 필드: 연락처 이름
  - ✓ phoneNumber 필드: 전화번호
- 생성자에서 필드 초기화에 필요한 모든 값을 받음
- 모든 필드에 대한 getter와 setter 메서드

ContactSet 클래스

코드 5-1 ch05_pjt_01\src\main\java\ch05_pjt_01\contact\ContactSet.java

```
01 package ch05_pjt_01.contact;
02
03 public class ContactSet {
04
05     private String name;
06     private String phoneNumber;
07
08     public ContactSet(String name, String phoneNumber) {
09         this.name = name;
10         this.phoneNumber = phoneNumber;
11     }
12
13     public String getName() { return name; }
14     public void setName(String name) { this.name = name; }
15
16     public String getPhoneNumber() { return phoneNumber; }
17     public void setPhoneNumber(String phoneNumber) {
18         this.phoneNumber = phoneNumber;
19     }
20 }
```

ContactDao 클래스

- 일반적인 DAO 클래스
  - ✓ 데이터베이스에 접속하고 Service에 의해서 호출됨
  - ✓ 데이터의 insert, search 기능을 수행함
- HashMap을 이용해서 데이터를 관리함
  - ✓ contactDB에서 key로 사용되는 값은 연락처 정보(ContactSet 클래스)에서 고윳값으로 부여되는 name을 이용함

■ ContactDao 클래스

코드 5-2 ch05_pjt_01\src\main\java\ch05_pjt_01\contact\dao\ContactDao.java

```
01 package ch05_pjt_01.contact.dao;
02
03 import java.util.HashMap;
04 import java.util.Map;
05
06 import ch05_pjt_01.contact.ContactSet;
07
08 public class ContactDao {
09
10     private Map<String, ContactSet> contactDB = new HashMap<String, ContactSet>();
11
12     public void insert(ContactSet contactSet) {
13         contactDB.put(contactSet.getName(), contactSet);
14     }
15
16     public ContactSet select(String name) {
17         return contactDB.get(name);
18     }
19
20     public Map<String, ContactSet> getContactDB() {
21         return contactDB;
22     }
23
24 }
```

2. 스마트폰 연락처 만들기

ContactRegisterService 클래스

- 연락처 정보를 데이터베이스에 저장하는 기능
- 추후 IoC 컨테이너에서 생성자에 ContactDao를 주입하게 됨
- verify() 메서드
  - ✓ 연락처를 등록하기 전에 name을 이용해서 기존에 등록된 연락처인지 판단하는 기능

ContactRegisterService 클래스

코드 5-3 ch05_pjt_01\src\main\java\ch05_pjt_01\contact\service\ContactRegisterService.java

```
01 package ch05_pjt_01.contact.service;
02
03 import ch05_pjt_01.contact.ContactSet;
04 import ch05_pjt_01.contact.dao.ContactDao;
05
06 public class ContactRegisterService {
07
08     private ContactDao contactDao;
09
10     public ContactRegisterService(ContactDao contactDao) {
11         this.contactDao = contactDao;
12     }
13
14     public void register(ContactSet contactSet) {
15         String name = contactSet.getName();
16         if (verify(name)) {
17             contactDao.insert(contactSet);
18         } else {
19             System.out.println("The name has already registered.");
20         }
21     }
22
23     public boolean verify(String name) {
24         ContactSet contactSet = contactDao.select(name);
25         return contactSet == null ? true : false;
26     }
27
28     public void setWordDao(ContactDao contactDao) {
29         this.contactDao = contactDao;
30     }
31
32 }
```

ContactSearchService 클래스

- ContactSearchService 클래스를 이용해서 특정 연락처의 정보를 얻을 수 있음
- Contact RegisterService와 마찬가지로 생성자에서 전달받은 DAO 객체를 contactDao 필드에 저장함
- searchContact() 메서드
  - ✓ DAO를 이용해서 데이터베이스에서 단어 정보를 가져옴

ContactSearchService 클래스

코드 5-4 ch05_pjt_01\src\main\java\ch05_pjt_01\contact\service\ContactSearchService.java

```
01 package ch05_pjt_01.contact.service;
02
03 import ch05_pjt_01.contact.ContactSet;
04 import ch05_pjt_01.contact.dao.ContactDao;
05
06 public class ContactSearchService {
07
08     private ContactDao contactDao;
09
10     public ContactSearchService(ContactDao contactDao) {
11         this.contactDao = contactDao;
12     }
13
14     public ContactSet searchContact(String name) {
15         if (verify(name)) {
16             return contactDao.select(name);
17         } else {
18             System.out.println("Contact information is available.");
19         }
20
21         return null;
22     }
23
24     public boolean verify(String name) {
25         ContactSet contactSet = contactDao.select(name);
26         return contactSet != null ? true : false;
27     }
28
29     public void setContactDao(ContactDao contactDao) {
30         this.contactDao = contactDao;
31     }
32
33 }
```

InitSampleData

- 프로그램 실행에 필요한 샘플 데이터
- 실제 데이터는 appCtx.xml에서 설정함

코드 5-5 ch05_pjt_01\src\main\java\ch05_pjt_01\contact\utils\InitSampleData.java

```
01 package ch05_pjt_01.contact.utils;
02
03 public class InitSampleData {
04
05     private String[] names;
06     private String[] phoneNumbers;
07
08     public String[] getNames() { return names; }
09
10     public void setNames(String[] names) { this.names = names; }
11
12     public String[] getPhoneNumbers() { return phoneNumbers; }
13
14     public void setPhoneNumbers(String[] phoneNumbers) {
15         this.phoneNumbers = phoneNumbers;
16     }
17
18 }
```

appCtx.xml

- [src/main/resources]에 객체를 생성하고 조립하는 appCtx.xml 생성함
- InitSampleData 빈을 생성하는 코드로 3개의 샘플 연락처를 이용함

코드 5-6 ch05_pjt_01\src\main\resources\appCtx.xml

```
01 <?xml version="1.0" encoding="UTF-8"?>
02
03 <beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd">
04
05     <bean id="initSampleData" class="ch05_pjt_01.contact.utils.InitSampleData">
06         <property name="names">
07             <array>
08                 <value>류현진</value>
09                 <value>손흥민</value>
10                 <value>김연경</value>
11             </array>
12         </property>
13         <property name="phoneNumbers">
14             <array>
15                 <value>010-0000-1111</value>
16                 <value>010-0000-2222</value>
17                 <value>010-0000-3333</value>
18             </array>
19         </property>
20     </bean>
21
```

■ DAO, Service, 빈을 생성하는 코드

코드 5-7 ch05_pjt_01\src\main\resources\appCtx.xml

```
01      <bean id="contactDao" class="ch05_pjt_01.contact.dao.ContactDao" />
02
03      <bean id="registerService" class="ch05_pjt_01.contact.service.ContactRegisterService">
04          <constructor-arg ref="contactDao" />
05      </bean>
06
07      <bean id="searchService" class="ch05_pjt_01.contact.service.ContactSearchService">
08          <constructor-arg ref="contactDao" />
09      </bean>
10
11  </beans>
```

■ DAO, Service, 빈을 생성하는 코드

- registerService와 searchService는 <constructor-arg/>를 이용해서 생성자에 의존 객체(DAO)를 주입받음

DAO

```
<bean id=contactDao class="ch05_pjt_01.contact.dao.ContactDao" />
```

의존 객체 주입

Service

```
<bean id="registerService" class="ch05_pjt_01.contact.service.ContactRegisaterService">
    <constructor-arg ref=contactDao />
</bean>

<bean id="searchService" class="ch05_pjt_01.contact.service.ContactSearchService">
    <constructor-arg ref=contactDao />
</bean>
```

그림 5-4 Service는 생성자에서 DAO를 주입받음

3. <context:annotation-config/>

■ DAO, Service, 빈을 생성하는 코드

- contactDao 의존 객체를 주입하기 위한 코드

```
<constructor-arg ref="contactDao" />
```

- 스프링의 '의존 객체 자동 주입' 기능을 이용하면 <constructor-arg/> 또는 <property/>를 이용해서 의존 객체를 주입할 필요 없이 자동으로 의존 객체가 주입됨

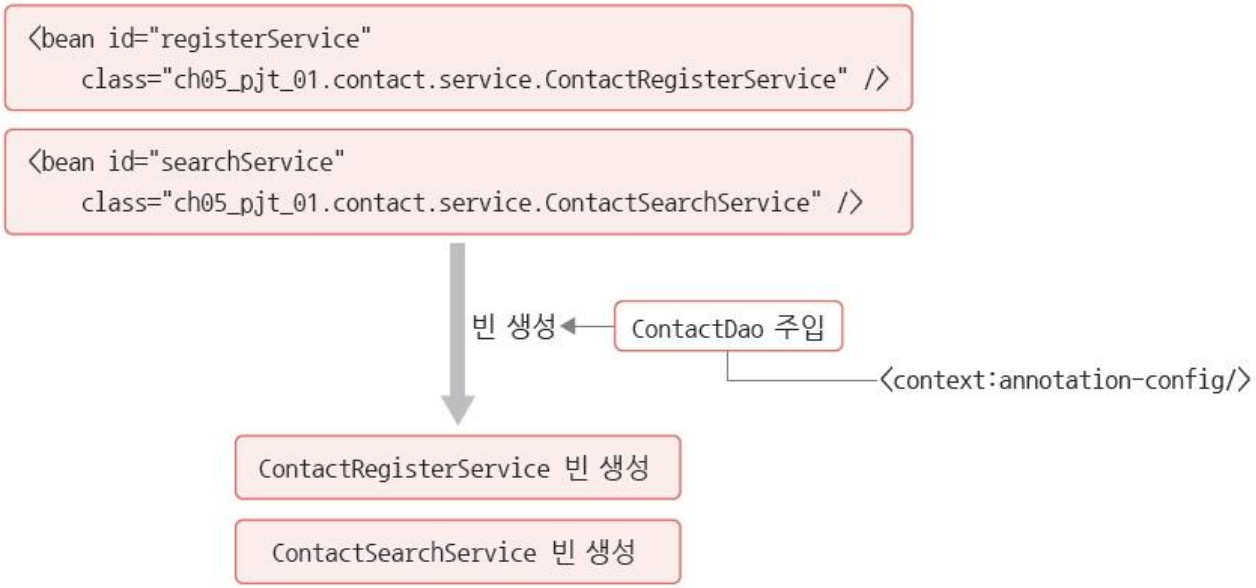


그림 5-5 <context:annotation-config/>로 인해서 ContactDao 자동 주입

3. <context:annotation-config/>

■ 자동으로 의존 객체 주입하기

- 1. context 네임스페이스와 스키마 추가하기
  - ✓ <context:annotation-config/>를 사용하기 위한 네임스페이스와 스키마를 appCtx.xml에 추가하기([코드 5-6]의 3행 수정)

```
<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">
```

- 2. <context:annotation-config/> 추가
  - ✓ 애너테이션을 사용하기 위한 <context:annotation-config/> 추가 ([코드 5-6]의 4행 다음 코드 추가)

```
<context:annotation-config/>
```

3. <context:annotation-config/>

■ 자동으로 의존 객체 주입하기

- 3. <constructor-arg> 제거
  - ✓ <[코드 5-7]의 4행, 8행 삭제

```
<bean id="registerService" class="ch05_pjt_01.contact.service.ContactRegisterService" />

<bean id="searchService" class="ch05_pjt_01.contact.service.ContactSearchService" />
```

  - ✓ <context:annotation-config/>에 의해서 <constructor-arg ref="contactDao" /> 없이도 생성자에 ContactDao가 자동으로 주입됨

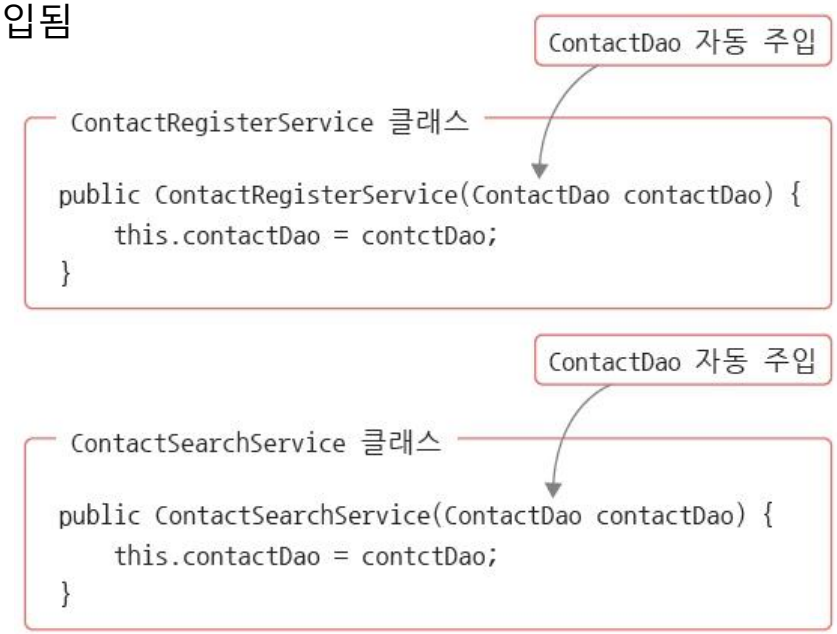


그림 5-6 ContactDao가 자동 주입됨

■ ContactDao 자동 주입 확인

- 1. MainClass 클래스

코드 5-8 ch05_pjt_01\src\main\java\ch05_pjt_01\contact\MainClass.java

```
01 package ch05_pjt_01.contact;
02
03 ...import문 생략...
04
05 public class MainClass {
06
07     public static void main(String[] args) {
08
09         // IoC 컨테이너 생성
10         GenericXmlApplicationContext ctx =
                   new GenericXmlApplicationContext("classpath:appCtx.xml");
11
12         // 샘플 데이터
13         InitSampleData initSampleData =
                   ctx.getBean("initSampleData", InitSampleData.class);
14         String[] names = initSampleData.getNames();
15         String[] phoneNumbers = initSampleData.getPhoneNumbers();
16
17         // 데이터 등록
18         ContactRegisterService registerService =
                   ctx.getBean("registerService", ContactRegisterService.class);
19         for (int i = 0; i < names.length; i++) {
20             ContactSet contactSet = new ContactSet(names[i], phoneNumbers[i]);
21             registerService.register(contactSet);
22         }
23
```

■ ContactDao 자동 주입 확인

```
24          // 데이터 조회
25          ContactSearchService searchService =
                    ctx.getBean("searchService", ContactSearchService.class);
26          ContactSet contactSet = searchService.searchContact("류현진");
27          System.out.println("name: " + contactSet.getName());
28          System.out.println("phone number: " + contactSet.getPhoneNumber());
29          System.out.println("----------------------------");
30
31          contactSet = searchService.searchContact("손흥민");
32          System.out.println("name: " + contactSet.getName());
33          System.out.println("phone number: " + contactSet.getPhoneNumber());
34          System.out.println("----------------------------");
35
36          contactSet = searchService.searchContact("김연경");
37          System.out.println("name: " + contactSet.getName());
38          System.out.println("phone number: " + contactSet.getPhoneNumber());
39          System.out.println("----------------------------");
40
41          ctx.close();
42      }
43  }
```

실행 결과

```
name: 류현진
phone number: 010-0000-1111
----------------------------
name: 손흥민
phone number: 010-0000-2222
----------------------------
name: 김연경
phone number: 010-0000-3333
```

■ ContactDao 자동 주입 확인

- 2. ContactRegisterService와 ContactSearchService의 생성자 내부에 contactDao를 출력하는 코드 추가

  ✓ 의존 객체 자동 주입을 더욱 확실하게 확인하기 위함

  ✓ ContactRegisterService 생성자

```
public ContactRegisterService(ContactDao contactDao) {
    System.out.println("contactDao: " + contactDao);  // contactDao 출력

    this.contactDao = contctDao;
}
```

  ✓ ContactSearchService 생성자

```
public ContactSearchService(ContactDao contactDao) {
    System.out.println("contactDao: " + contactDao);  // contactDao 출력

    this.contactDao = contctDao;
}
```

실행 결과

```
contactDao: ch05_pjt_01.contact.dao.ContactDao@21282ed8
contactDao: ch05_pjt_01.contact.dao.ContactDao@21282ed8
```

메모리 주소 값으로 실행할 때마다 다르게 나올 수 있음

Section 02

@Autowired를 이용한 의존 객체 자동 주입

1. 생성자가 여러 개인 경우

■ 생성자가 오버로딩되어 여러 개가 있는 경우

- Q) 의존 객체는 어떤 생성자에 자동 주입될까?
  - ✓ 이를 확인하기 위해 Service 클래스에 디폴트 생성자를 추가
- ContactRegisterService 클래스 생성자

```
public ContactRegisterService() {
    System.out.println("default constructor");
}

public ContactRegisterService(ContactDao contactDao) {
    System.out.println("contactDao: " + contactDao);   // contactDao 출력

    this.contactDao = contactDao;
}
```

디폴트 생성자 추가

- ContactSearchService 클래스 생성자

```
public ContactSearchService() {
    System.out.println("default constructor");
}

public ContactSearchService(WordDao wordDao) {
    System.out.println("contactDao: " + contactDao);   // contactDao 출력

    this.contactDao = contactDao;
}
```

디폴트 생성자 추가

■ 생성자가 오버로딩되어 여러 개가 있는 경우

- 실행 결과 → 디폴트 생성자가 호출됨
  - ✓ appCtx.xml에서 <bean> 태그에 <constructor-arg/>를 사용하지 않았기 때문

실행 결과

```
default constructor
default constructor
Exception in thread "main" java.lang.NullPointerException
        at ch05_pjt_01.word.service.ContactRegisterService.verify(ContactRegisterService.java:30)
        at ch05_pjt_01.word.service.ContactRegisterService.register(ContactRegisterService.java:22)
        at ch05_pjt_01.word.MainClass.main(MainClass.java:34)
```

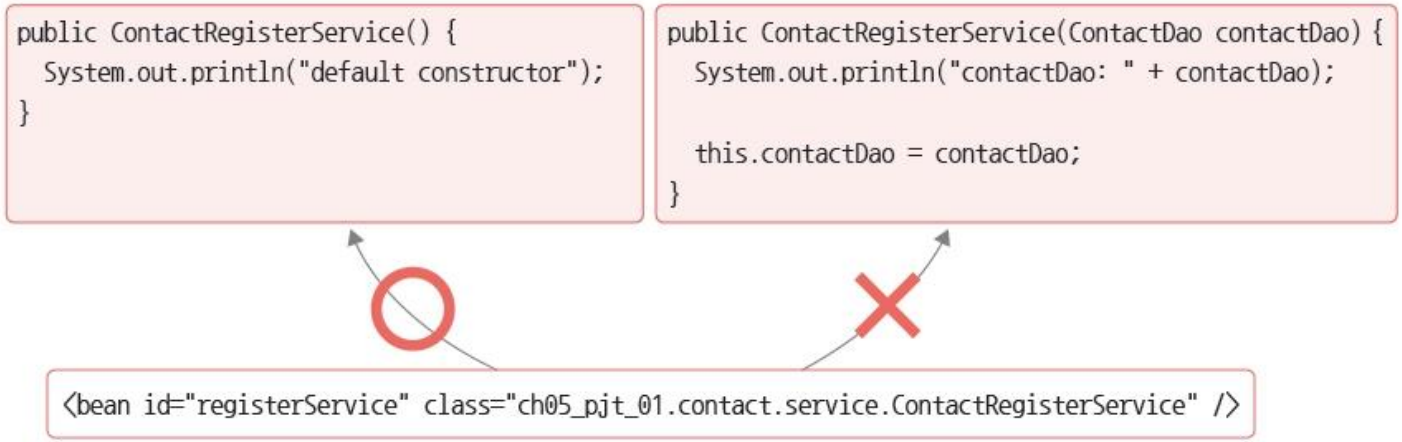


그림 5-7 디폴트 생성자 호출로 빈을 생성함

1. 생성자가 여러 개인 경우

■ 생성자가 오버로딩되어 여러 개가 있는 경우

- 실행 결과 → 디폴트 생성자가 호출됨
  - ✓ 디폴트 생성자가 호출됐기 때문에 contactDao 필드는 초기화하지 못함
  - ✓ 결과적으로 register()와 verify() 호출 시 NullPointerException이 발생

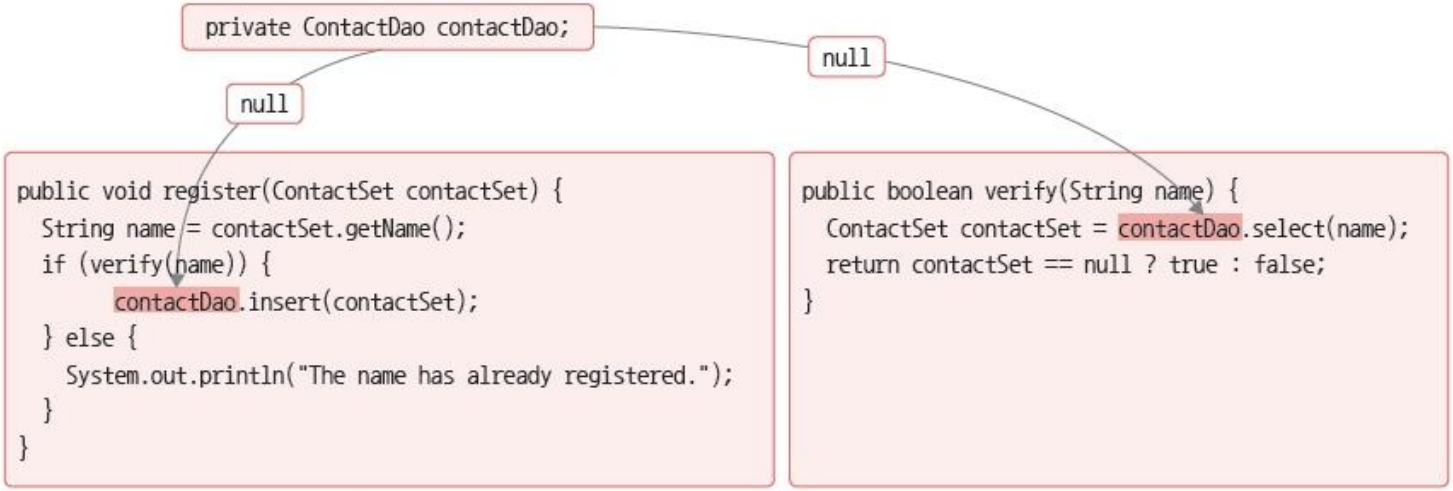


그림 5-8 contactDao가 null이어서 regist()와 verify()에서 contactDao를 참조할 때 NullPointerException이 발생함

1. 생성자가 여러 개인 경우

■ 생성자가 오버로딩되어 여러 개가 있는 경우

- Q) ContactDao가 Service 객체에 자동 주입될 수 있는 방법은?
- A) ContactDao를 필요로 하는 곳에 @Autowired 애너테이션을 명시하기

- ContactRegisterService 클래스 생성자

```
import org.springframework.beans.factory.annotation.Autowired;

public ContactRegisterService() {
    System.out.println("default constructor");
}

@Autowired
public ContactRegisterService(ContactDao contactDao) {
    System.out.println("contactDao: " + contactDao);  // contactDao 출력

    this.contactDao = contactDao;
}
```

1. 생성자가 여러 개인 경우

■ 생성자가 오버로딩되어 여러 개가 있는 경우

- ContactSearchService 클래스 생성자

```
import org.springframework.beans.factory.annotation.Autowired;

public ContactSearchService() {
    System.out.println("default constructor");
}

@Autowired
public ContactSearchService(ContactDao contactDao) {
    System.out.println("contactDao: " + contactDao);  // contactDao 출력

    this.contactDao = contactDao;
}
```

실행 결과

```
contactDao: ch05_pjt_01.contact.dao.WordDao@4686afc2
contactDao: ch05_pjt_01.contact.dao.WordDao@4686afc2
...생략...
```

메모리 주소 값으로 실행할 때마다 다르게 표시될 수 있음

1. 생성자가 여러 개인 경우

■ 생성자가 오버로딩되어 여러 개가 있는 경우

- 실행 결과 → @Autowired를 이용해서 WordDao가 생성자에 정상적으로 자동 주입됨

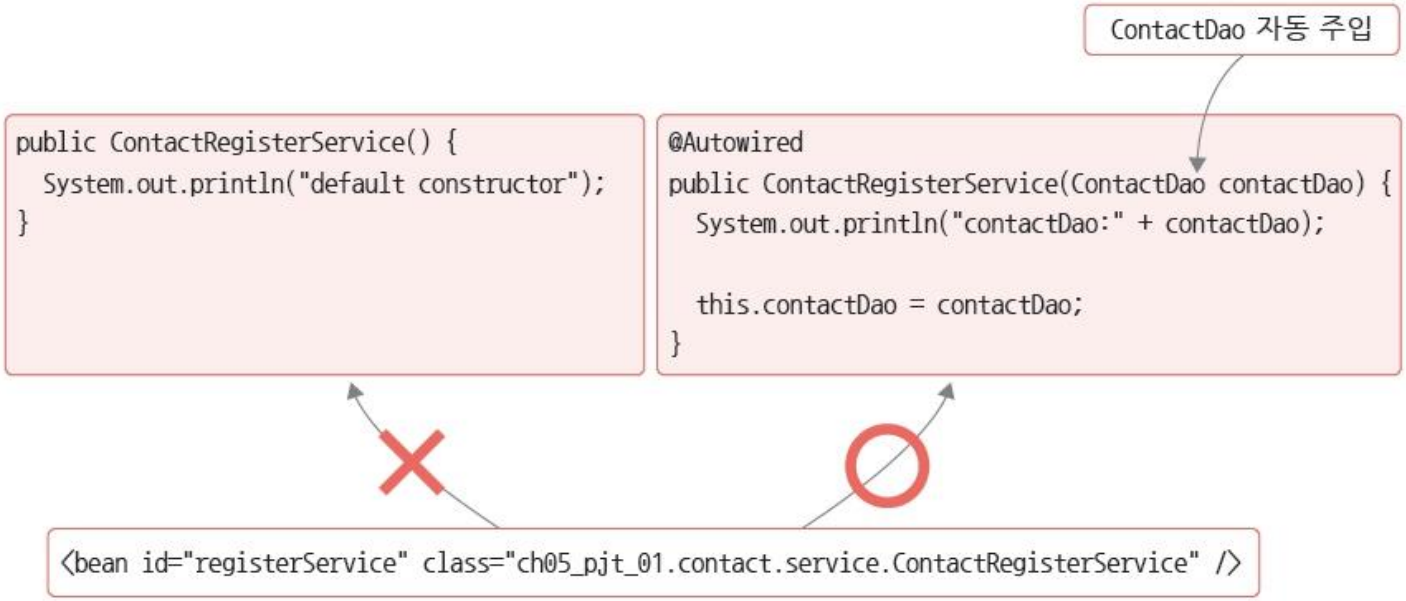


그림 5-9 @Autowired를 이용한 생성자 자동 주입

1. 생성자가 여러 개인 경우

■ 생성자가 오버로딩되어 여러 개가 있는 경우

- 정리
  - ✓ @Autowired 애너테이션을 사용하면 객체가 생성될 때 데이터 타입을 검색해서 알맞은 객체를 주입함
  - ✓ 즉, ContactRegisterService 객체가 생성될 때 필요한 ContactDao 객체를 데이터 타입으로 검색해서 알맞은 객체를 자동으로 주입함
  - ✓ ContactSearch Service의 경우에도 마찬가지로 필요한 의존 객체 타입을 검색하여 객체를 자동으로 주입함

2. @Autowired 적용 대상

■ 필드에 @Autowired 적용

- ContactRegisterService와 ContactSearchService에 생성자가 아닌 contactDao 필드에 @Autowired를 명시해서 의존 객체를 자동 주입

- ContactRegisterService 클래스

```
import org.springframework.beans.factory.annotation.Autowired;

@Autowired
private ContactDao contactDao;

public ContactRegisterService() {
    System.out.println("default constructor");
}

//    @Autowired
//    public ContactSearchService(ContactDao contactDao) {
//        System.out.println("contactDao: " + contactDao);
//
//        this.contactDao = contactDao;
//    }
```

2. @Autowired 적용 대상

■ 필드에 @Autowired 적용

- ContactSearchService 클래스

```
import org.springframework.beans.factory.annotation.Autowired;

@Autowired
private ContactDao contactDao;

public ContactSearchService() {
    System.out.println("default constructor");
}

//    @Autowired
//    public ContactSearchService(ContactDao contactDao) {
//        System.out.println("contactDao: " + contactDao);
//
//        this.contactDao = contactDao;
//    }
```

실행 결과

```
default constructor
default constructor
...생략...
```

디폴트 생성자 호출됨

2. @Autowired 적용 대상

■ 메서드에 @Autowired 적용

- @Autowired를 메서드에 적용할 수도 있음
- setter 메서드를 만들고 @Autowired를 적용해보기
  - ✓ setter 메서드를 이용할 때도 기본적으로 객체가 생성되고 setter 메서드를 이용해서 의존 객체가 주입됨

- ContactRegisterService 클래스

```
import org.springframework.beans.factory.annotation.Autowired;

//    @Autowired
//    private ContactDao contactDao;
//    public ContactRegisterService() {
//        System.out.println("default constructor");
//    }

//    @Autowired
//    public ContactSearchService(ContactDao contactDao) {
//        System.out.println("contactDao: " + contactDao);
//
//        this.contactDao = contactDao;
//    }

    @Autowired
    public void setContactDao(ContactDao contactDao) {
        this.contactDao = contactDao;
    }
```

2. @Autowired 적용 대상

■ 메서드에 @Autowired 적용

- ContactSearchService 클래스

```
import org.springframework.beans.factory.annotation.Autowired;

//    @Autowired
//private ContactDao contactDao;

//    public ContactSearchService() {
//        System.out.println("default constructor");
//    }

//    @Autowired
//    public ContactSearchService(ContactDao wordDao) {
//        System.out.println("contactDao: " + contactDao);
//
//        this.contactDao = contactDao;
//    }

    @Autowired
    public void setContactDao(ContactDao contactDao) {
        this.contactDao = contactDao;
    }
```

실행 결과

```
name: 류현진
phone number: 010-0000-1111
-------------------------
name: 손흥민
phone number: 010-0000-2222
-------------------------
name: 김연경
phone number: 010-0000-3333
-------------------------
```

Section 03

@Resource를 이용한 의존 객체 자동 주입

1. @Autowired와 @Resource의 차이점

■ @Resource와 @Autowired의 차이점

- @Autowired
  - ✓ 적합한 데이터 타입을 이용해서 의존 객체를 자동으로 주입함
- @Resource
  - ✓ 빈의 이름을 이용해서 의존 객체를 자동으로 주입함

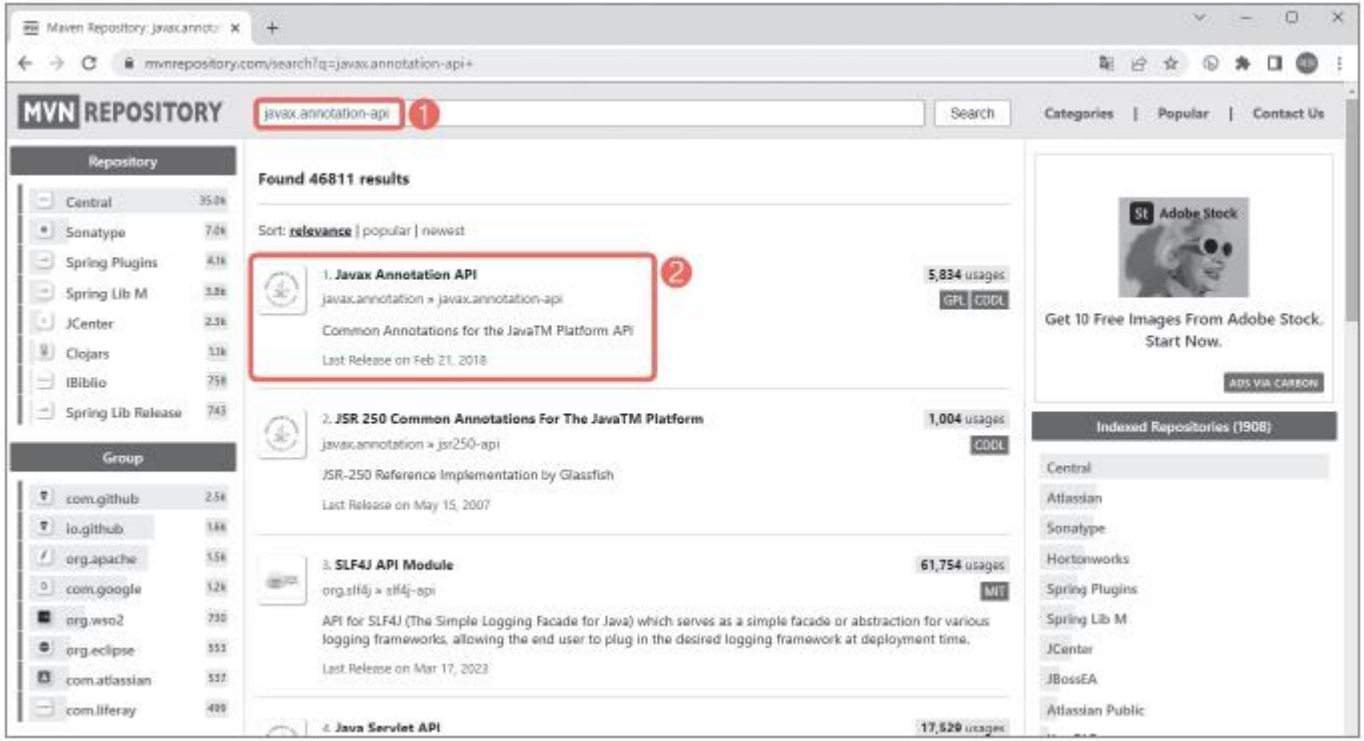
표 5-1 @Autowired와 @Resource의 비교

@Autowired	@Resource
• 데이터 타입을 이용한 의존 객체 자동 주입 • 필드, 생성자, 메서드에 사용 가능	• 빈 객체의 이름을 이용한 의존 객체 자동 주입 • 필드, 메서드에 사용 가능(생성자 사용 불가능)

2. @ Resource 적용 대상

■ 필드에 @ Resource 적용

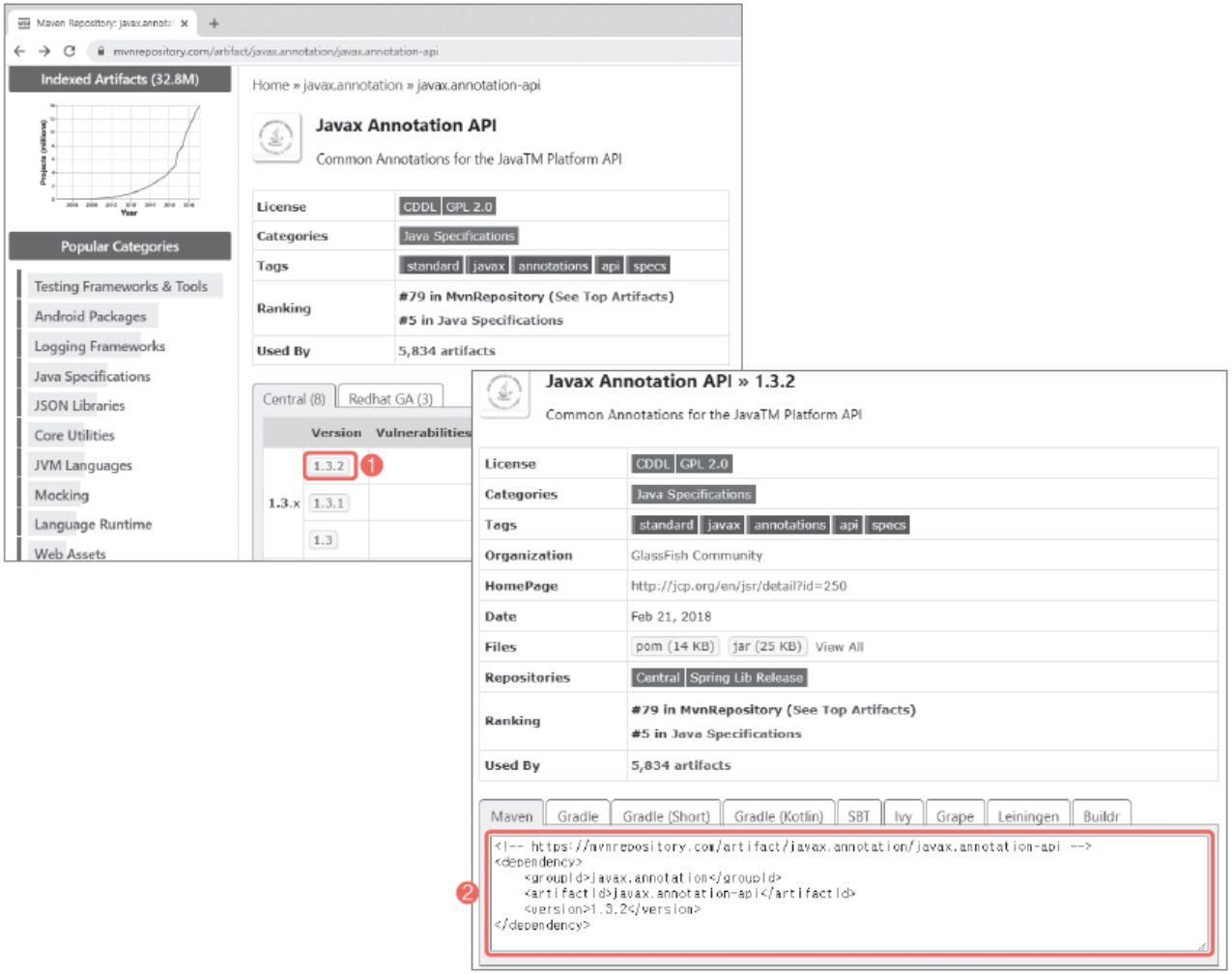
- @Resource는 스프링이 아닌 자바에서 제공하는 애너테이션으로 관련 모듈을 pom.xml에 설정해야 함
- 1. 메인 리포지터리(https://mvnrepository.com/)에 접속하여 javax.annotation-api 모듈을 검색하고, Javax Annotation API를 선택



2. @ Resource 적용 대상

■ 필드에 @ Resource 적용

- 2. Javax AnnotationAPI 페이지에서 1.3.2를 클릭하고, 메이븐 코드를 복사



2. @ Resource 적용 대상

■ 필드에 @ Resource 적용

- 4. 복사한 코드를 pom.xml에 붙여넣기

```
<!-- spring-context 모듈-->
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.2.9.RELEASE</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/javax.annotation/javax.annotation-api -->
    <dependency>
        <groupId>javax.annotation</groupId>
        <artifactId>javax.annotation-api</artifactId>
        <version>1.3.2</version>
    </dependency>
</dependencies>
```

2. @ Resource 적용 대상

■ 필드에 @ Resource 적용

- 5. @Resource를 이용해서 ContactRegisterService와 ContactSearchService에서 ContactDao를 필드에 자동 주입

- ContactRegisterService와 ContactSearchService 클래스
  - ✓ 이때 앞에서 작성한 @Autowired는 주석 처리

```
import javax.annotation.Resource;

@Resource          //@Autowired는 주석 처리
private ContactDao contactDao;
```

실행 결과

```
name: 류현진
phone number: 010-0000-1111
-----------------------------
name: 손흥민
phone number: 010-0000-2222
-----------------------------
name: 김연경
phone number: 010-0000-3333
-----------------------------
```

2. @ Resource 적용 대상

■ 메서드에 @Resource 적용

- @Resource를 메서드에 적용할 수도 있음

- ContactRegisterService와 ContactSearchService 클래스

```
import javax.annotation.Resource;

@Resource
public void setContactDao(ContactDao contactDao) {
    this.contactDao = contactDao;
}
```

실행 결과

```
name: 류현진
phone number: 010-0000-1111
-----------------------------
name: 손흥민
phone number: 010-0000-2222
-----------------------------
name: 김연경
phone number: 010-0000-3333
-----------------------------
```

3. 다수의 빈 객체 중 의존 대상 객체 선택 방법

■ 자동 주입 대상 객체가 2개 이상인 경우

- 에러 발생함
  - ✓ appCtx.xml에 다음 두 줄을 추가하여 프로젝트를 실행해보기

```
<bean id="contactDao1" class="ch05_pjt_01.contact.dao.ContactDao" />
<bean id="contactDao2" class="ch05_pjt_01.contact.dao.ContactDao" />
<bean id="contactDao3" class="ch05_pjt_01.contact.dao.ContactDao" />

<bean id="registerService" class="ch05_pjt_01.contact.service.ContactRegisterService" />

<bean id="searchService" class="ch05_pjt_01.contactservice.ContactSearchService" />
```

추가된 코드

실행 결과

```
WARNING: Exception encountered during context initialization - cancelling refresh attempt:
org.springframework.beans.factory.UnsatisfiedDependencyException: Error creating bean with
name 'registerService': Unsatisfied dependency expressed through method 'setContactDao'
parameter 0; nested exception is
```

3. 다수의 빈 객체 중 의존 대상 객체 선택 방법

■ 자동 주입 대상 객체가 2개 이상인 경우

- 실행 결과 → 에러 발생
  - ✓ 스프링은 자동으로 주입되는 의존 객체 대상이 3개나 되지만 어떤 의존 객체가 주입 대상인지 찾지 못함
- 동일한 타입의 의존 객체가 2개 이상인 경우에는 어떤 의존 객체를 자동 주입해야 하는지 판단할 수가 없기 때문
- 해결 방법
  - ✓ 의존 대상 객체를 개발자가 지정해주기
  - ✓ appCtx.xml의 다음 세 줄 수정하기

```
<bean id="contactDao1" class="ch05_pjt_01.contact.dao.ContactDao" />
<bean id="contactDao2" class="ch05_pjt_01.contact.dao.ContactDao" />
<bean id="contactDao3" class="ch05_pjt_01.contact.dao.ContactDao" />
```

```
<bean id="contactDao1" class="ch05_pjt_01.contact.dao.ContactDao">
    <qualifier value="usedDao"/>
</bean>
<bean id="contactDao2" class="ch05_pjt_01.contact.dao.ContactDao" />
<bean id="contactDao3" class="ch05_pjt_01.contact.dao.ContactDao" />
```

3. 다수의 빈 객체 중 의존 대상 객체 선택 방법

■ 자동 주입 대상 객체가 2개 이상인 경우

- <qualifier> 태그 추가
  - ✓ 의존 대상 객체로 지정한다는 의미
  - ✓ 의존 객체가 자동 주입되는 ContactRegisterService와 ContactSearchService에 @Qualifier 애너테이션을 추가해주어야 함
  - ✓ 이때 @Autowired와 @Resource 중 어느 쪽을 사용하든 상관 없음
- @Autowired와 @Qualifier를 이용한 특정 의존 객체 지정

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;

@Autowired
@Qualifier("usedDao")
private ContactDao contactDao;
```

실행 결과

```
name: 류현진
phone number: 010-0000-1111
-----------------------------
name: 손흥민
phone number: 010-0000-2222
-----------------------------
name: 김연경
phone number: 010-0000-3333
-----------------------------
```

- @Resource와 @Qualifier를 이용한 특정 의존 객체 지정

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;

@Autowired
@Qualifier("usedDao")
private ContactDao contactDao;
```

3. 다수의 빈 객체 중 의존 대상 객체 선택 방법

■ 생성자 또는 멤버의 이름이 빈의 id와 같은 경우

- <qualifier>와 @Qualifier를 이용하지 않아도 의존 객체가 자동으로 주입됨

- appCtx.xml

```
<bean id="contactDao" class="ch05_pjt_01.contact.dao.ContactDao" />
<bean id="contactDao2" class="ch05_pjt_01.contact.dao.ContactDao" />
<bean id="contactDao3" class="ch05_pjt_01.contact.dao.ContactDao" />
```

- ContactRegisterService와 ContactSearchService 클래스

```
@Autowired
private ContactDao contactDao;
```

실행 결과

```
name: 류현진
phone number: 010-0000-1111
-------------------------------
name: 손흥민
phone number: 010-0000-2222
-------------------------------
name: 김연경
phone number: 010-0000-3333
-------------------------------
```

3. 다수의 빈 객체 중 의존 대상 객체 선택 방법

■ 매개변수 개수가 2개 이상인 경우

- 주입 대상 매개변수에 @Qualifier를 적용하면 의존 객체 자동 주입은 가능함

- appCtx.xml

```
<bean id="firstBean1" class="ch05_pjt_01.contact.service.FirstBean">
    <qualifier value="usedBean" />
</bean>
<bean id="firstBean2" class="ch05_pjt_01.contactservice.FirstBean"/>
<bean id="firstBean3" class="ch05_pjt_01.contact.service.FirstBean"/>

<bean id="secondBean" class="ch05_pjt_01.contactservice.SecondBean"/>

<bean id="thirdBean" class="ch05_pjt_01.contact.service.ThirdBean"/>

<bean id="fourthBean1" class="ch05_pjt_01.contact.service.FourthBean">
    <qualifier value="usedBean" />
</bean>
<bean id="fourthBean2" class="ch05_pjt_01.contact.service.FourthBean"/>
<bean id="fourthBean3" class="ch05_pjt_01.contact.service.FourthBean"/>

<bean id="autoWiredEx" class="ch05_pjt_01.contact.service.AutoWiredEx"/>
```

3. 다수의 빈 객체 중 의존 대상 객체 선택 방법

■ 매개변수 개수가 2개 이상인 경우

- AutoWiredEx.java

```
public class AutoWiredEx {

    @Autowired
    public AutoWiredEx(@Qualifier("usedBean") FirstBean fBean, SecondBean sBean) {
        System.out.println("fBean: " + fBean);
        System.out.println("sBean: " + sBean);
    }

    @Autowired
    public void autowiredMethod(ThirdBean tBean, @Qualifier("usedBean") FourthBean fBean) {
        System.out.println("tBean: " + tBean);
        System.out.println("fBean: " + fBean);
    }

}
```

실행 결과

```
fBean: ch05_pjt_01.contact.service.FirstBean@46cdf8bd
sBean: ch05_pjt_01.contact.service.SecondBean@f0c8a99
tBean: ch05_pjt_01.contact.service.ThirdBean@6892b3b6
fBean: ch05_pjt_01.contact.service.FourthBean@6e6f2380
```

4. 의존 객체 자동 주입 시 필수 여부 지정하기

■ 빈을 생성하지 않고 @Autowired를 사용한 경우

- 의존 객체를 자동으로 주입하는 과정에서 의존 객체를 찾을 수 없어 에러 발생

- appCtx.xml

```
<!-- <bean id="contactDao" class="ch05_pjt_01.contact.dao.WordDao" /> -->

<bean id="registerService" class="ch05_pjt_01.contact.service.ContactRegisterService" />

<bean id="searchService" class="ch05_pjt_01.contact.service.ContactearchService" />
```

- ContactRegisterService와 ContactSearchService 클래스

```
@Autowired
private ContactDao contactDao;
```

실행 결과

```
Exception in thread "main" org.springframework.beans.factory.UnsatisfiedDependencyException:
Error creating bean with name 'registerService'
Caused by: org.springframework.beans.factory.NoSuchBeanDefinitionException: No qualifying
bean of type 'ch05_pjt_01.contact.dao.ContactDao'
```

4. 의존 객체 자동 주입 시 필수 여부 지정하기

■ 빈을 생성하지 않고 @Autowired를 사용한 경우

- required 속성
  - ✓ @Aurowired에 required 속성을 false로 지정하면 의존 객체 자동 주입이 필수가 아닌 필요에 따라서만 주입됨
  - ✓ 따라서 의존 대상 객체를 못 찾아서 발생하는 에러를 피할 수 있음

```
@Autowired(required = false)
private ContactDao contactDao;
```

Section 04

@Inject를 이용한 의존 객체 자동 주입

1. @Inject를 이용한 의존 객체 자동 주입

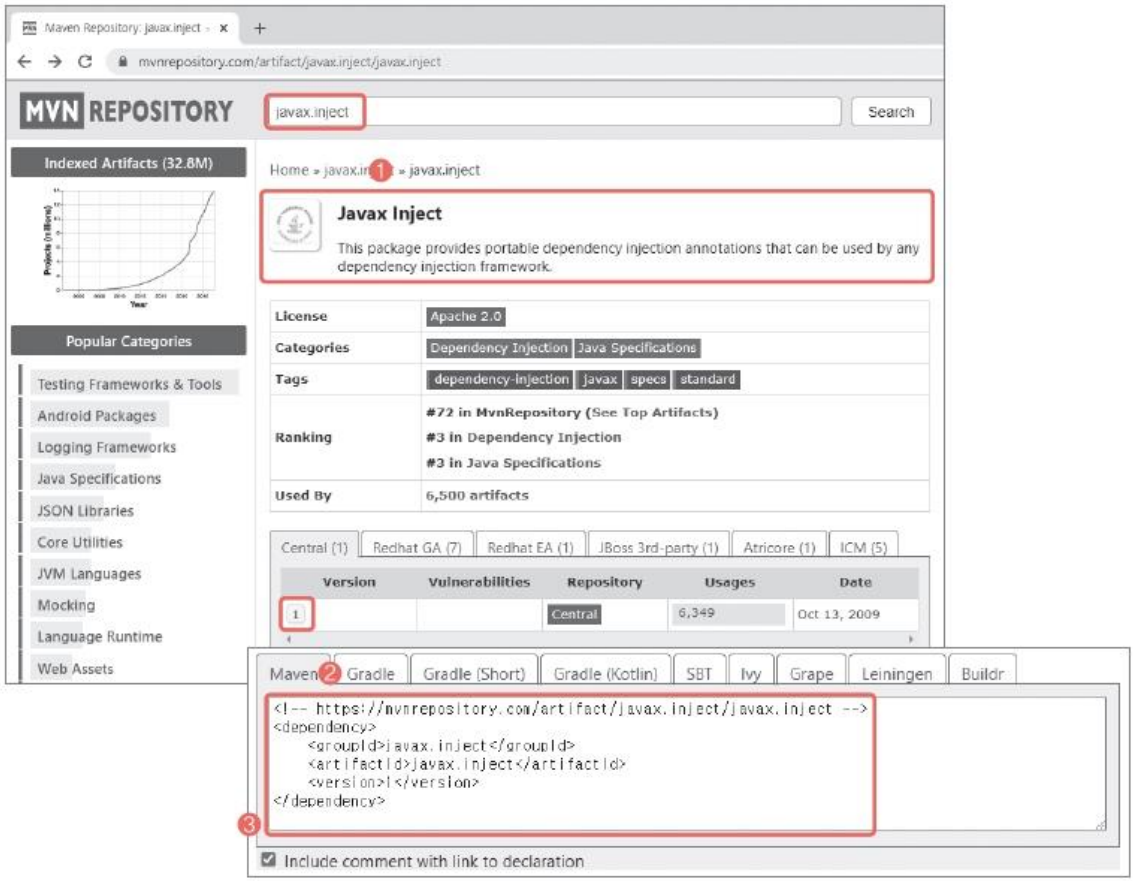
■ @Autowired와 @Inject의 차이점

- @Autowired
  - ✓ required 속성을 이용해서 의존 객체가 없어도 에러를 피할 수 있음
- @Inject
  - ✓ required 속성을 지원하지 않음
  - ✓ 사용 방법은 @Autowired와 동일하지만, @Inject는 @Autowired와 달리 스프링이 아닌 자바에서 제공하는 애너테이션으로 관련 모듈을 pom.xml에 설정해야 함
- @Autowired와 동일하게 @Inject 애너테이션을 이용해서 의존 객체를 자동으로 주입을 할 수 있음

1. @Inject를 이용한 의존 객체 자동 주입

■ @Inject를 이용한 의존 객체 자동 주입

- 메인 리포지터리(https://mvnrepository.com/)에 접속하여 javax.inject 모듈 검색
- 1을 클릭하여 메이븐 코드를 복사해 pom.xml에 붙여넣기



■ @Inject를 이용한 의존 객체 자동 주입

- pom.xml

```
<!-- spring-context 모듈-->
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.2.9.RELEASE</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/javax.inject/javax.inject -->
    <dependency>
        <groupId>javax.inject</groupId>
        <artifactId>javax.inject</artifactId>
        <version>1</version>
    </dependency>
</dependencies>
```

■ @Inject를 이용한 의존 객체 자동 주입

- ContactRegisterService와 ContactSearchService 클래스

```
<!-- 필드 자동 주입 -->
@Inject
private ContactDao contactDao;

<!-- 생성자 자동 주입 -->
@Inject
public ContactRegisterService(ContactDao contactDao) {
    this.contactDao = wordcontactDao;
}

@Inject
public ContactSearchService(ContactDao contactDao) {
    this.contactDao = wordcontactDao;
}

<!-- 메서드 자동 주입 -->
@Inject
public void setContactDao(ContactDao contactDao) {
    this.contactDao = wordcontactDao;
}
```

ContactRegisterService 클래스

ContactSearchService 클래스

실행 결과

```
name: 류현진
phone number: 010-0000-1111
-----------------------------
name: 손흥민
phone number: 010-0000-2222
-----------------------------
name: 김연경
phone number: 010-0000-3333
-----------------------------
```

1. @Inject를 이용한 의존 객체 자동 주입

■ 자동 주입 대상 객체가 2개 이상인 경우

- @Named 애너테이션 이용
  - ✓ 특정 객체를 지정할 수 있음
  - ✓ 또한 @Autowired처럼 특정 객체를 지정하지 않으면 에러가 발생함
- appCtx.xml

```
<bean id="contactDao1" class="ch05_pjt_01.contact.dao.ContactDao" />
<bean id="contactDao2" class="ch05_pjt_01.contact.dao.ContactDao" />
<bean id="contactDao3" class="ch05_pjt_01.contactdao.ContactDao" />

<bean id="registerService" class="ch05_pjt_01.contact.service.ContactRegisterService" />

<bean id="searchService" class="ch05_pjt_01.contact.service.ContactSearchService" />
```

1. @Inject를 이용한 의존 객체 자동 주입

■ 자동 주입 대상 객체가 2개 이상인 경우

- ContactRegisterService와 ContactSearchService 클래스

```
<!-- 필드 자동 주입 -->
@Inject
@Named("contactDao1")
private ContactDao contactDao;

<!-- 생성자 자동 주입 -->
@Inject
@Named("contactDao1")
public ContactRegisterService(ContactDao contactDao) {
    this.contactDao = contactDao;
}

@Inject
@Named("contactDao1")
public ContactSearchService(ContactDao contactDao) {
    this.contactDao  = contactDao;
}

<!--메서드 자동 주입-->
@Inject
@Named("contactDao1")
public void setContactDao(ContactDao contactDao) {
    this.contactDao = contactDao;
}
```

ContactRegisterService 클래스

ContactSearchService 클래스

1. @Inject를 이용한 의존 객체 자동 주입

■ 디폴트 생성자가 없어서 에러가 발생하는 경우

- 필드 또는 메서드에 @Inject를 적용할 때 반드시 디폴트 생성자가 있어야 정상적으로 빈이 생성되고 의존 객체가 자동 주입될 수 있음

- ContactRegisterService

```
@Inject
@Named("contactDao1")
private ContactDao contactDao;

public ContactRegisterService(ContactDao contactDao) {
    this.contactDao = contactDao;
}
```

1. @Inject를 이용한 의존 객체 자동 주입

■ 디폴트 생성자가 없어서 에러가 발생하는 경우

- ContactSearchService 클래스

```
@Inject
@Named("contactDao1")
private ContactDao contactDao;

public ContactSearchService(ContactDao contactDao) {
    this.contactDao = contactDao;
}
```

실행 결과

```
Exception in thread "main"
org.springframework.beans.factory.UnsatisfiedDependencyException: Error creating bean
```

빈 생성 오류