

02강 스프링 DI

예제 준비

다음의 정보로 메이븐 프로젝트를 생성하자

- Group ID : com.green
- Artifact ID : ex02

POM.xml에 다음 내용을 추가한 다음 메이븐 업데이트를 진행한다

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>4.1.0.RELEASE</version>
  </dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.1</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
        <encoding>utf-8</encoding>
      </configuration>
    </plugin>
  </plugins>
</build>
```

예제 준비

-spring.exception

- AlreadyExistingMemberException.java
- IdPasswordNotMatchingException.java
- MemberNotFoundException.java

-spring.vo

- Member.java
- RegisterRequest.java

-spring.dao

- MemberDao.java

-spring.service

- MemberRegisterService.java
- ChangePasswordService.java

-spring.main

예제 준비

-spring.exception

- AlreadyExistingMemberException.java

```
public class AlreadyExistingMemberException extends RuntimeException{  
    public AlreadyExistingMemberException(String message) {  
        super(message);  
    }  
}
```

예제 준비

-spring.exception

- IdPasswordNotMatchingException.java

```
public class IdPasswordNotMatchingException extends RuntimeException{  
  
}
```

-spring.exception

- MemberNotFoundException.java

```
public class MemberNotFoundException extends RuntimeException{  
  
}
```

예제 준비

-spring.vo

- Member.java

```
import java.util.Date;
import spring.exception.IdPasswordNotMatchingException;

public class Member {
    private Long id;
    private String email;
    private String password;
    private String name;
    private Date registerDate;

    public Member(String email, String password, String name, Date registerDate) {
        this.email = email;
        this.password = password;
        this.name = name;
        this.registerDate = registerDate;
    }

    public void changePassword(String oldPassword, String newPassword) {
        if (!password.equals(oldPassword))
            throw new IdPasswordNotMatchingException();
        this.password = newPassword;
    }
}
```

=> 나머지는 Getter,Setter메서드로 채운다

예제 준비

-spring.vo

- RegisterRequest.java

```
public class RegisterRequest {  
  
    private String email;  
    private String password;  
    private String confirmPassword;  
    private String name;  
  
    public boolean isPasswordEqualToConfirmPassword() {  
        return password.equals(confirmPassword);  
    }  
}
```

=> 나머지는 Getter,Setter메서드로 채운다

예제 준비

-spring.dao

- MemberDao.java

```
public class MemberDao {  
    private static long nextId = 0;  
  
    private Map<String, Member> map = new HashMap<>();  
  
    public Member selectByEmail(String email) {  
        return map.get(email);  
    }  
  
    public void insert(Member member) {  
        member.setId(++nextId);  
        map.put(member.getEmail(), member);  
    }  
  
    public void update(Member member) {  
        map.put(member.getEmail(), member);  
    }  
  
    public Collection<Member> selectAll() {  
        return map.values();  
    }  
}
```


예제 준비

-spring.service

- MemberRegisterService.java

```
public class MemberRegisterService {  
  
    private MemberDao memberDao = new MemberDao();  
  
    public void regist(RegisterRequest req) {  
        Member member = memberDao.selectByEmail(req.getEmail());  
        if (member != null) {  
            throw new AlreadyExistingMemberException("dup email " + req.getEmail());  
        }  
        Member newMember = new Member(  
            req.getEmail(), req.getPassword(), req.getName(),  
            new Date());  
        memberDao.insert(newMember);  
    }  
}
```

예제 준비

-spring.service

- ChangePasswordService.java

```
public class ChangePasswordService {  
  
    private MemberDao memberDao = new MemberDao();  
  
    public void changePassword(String email, String oldPwd, String newPwd) {  
        Member member = memberDao.selectByEmail(email);  
        if (member == null)  
            throw new MemberNotFoundException();  
  
        member.changePassword(oldPwd, newPwd);  
  
        memberDao.update(member);  
    }  
}
```

의존이란?

- DI는 Dependency Injection의 약자로 '의존 주입' 이라고 해석된다.
- 여기서 의존이란 객체간의 의존을 의미한다.

```
public class MemberRegisterService {  
    private MemberDao memberDao = new MemberDao();  
  
    public void regist(RegisterRequest req) {  
        Member member = memberDao.selectByEmail(req.getEmail());  
        if (member != null) {  
            throw new AlreadyExistingMemberException("dup email " + req.getEmail());  
        }  
        Member newMember = new Member(  
            req.getEmail(), req.getPassword(), req.getName(),  
            new Date());  
        memberDao.insert(newMember);  
    }  
}
```

- 여기서 눈여겨 봐야 할 부분은 MemberRegisterService 클래스가 DB 처리를 위해서 memberDao의 메소드를 사용한다는 점이다.
=> regist메서드를 동작시키려면 반드시 memberDao클래스의 메서드가 필요하다
- 이렇게 한 클래스가 동작하기 위해서 반드시 다른 클래스를 필요로 하는 것
이것을 '의존한다'라고 표현한다.

DI를 통한 의존 처리

- MemberRegisterService 클래스가 memberDao 클래스에 의존한다고 볼 수 있다
- 이때 의존하는 대상이 있다면 그 대상을 구하는 방법이 필요하게 되는데
- 첫번째 방법은 의존 대상을 '직접 생성'하는 것이다.
 - `private MemberDao memberdao = new MemberDao();`
- MemberRegisterService의 객체가 생성된다면 내부적으로 MemberDao객체가 자동으로 함께 생성된다.
- 다만 이렇게 직접 생성하는 경우 유지보수의 관점에서 문제점이 생길 수 있다

DI를 통한 의존 처리

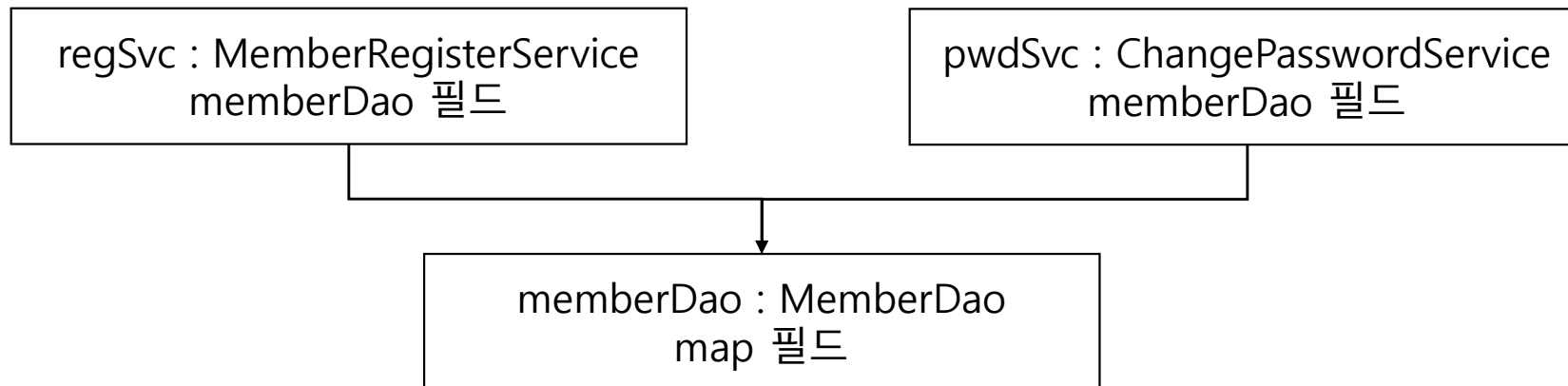
- 의존 객체를 구하는 두번째 방법 – DI(의존 주입)이다.
- 의존 주입은 의존하는 객체를 직접 생성하지 않고 전달 받는 방식을 사용한다.
- 앞서 살펴본 예시를 다음과 같이 바꿔 본다.
 - 예시2

```
public class MemberRegisterService {  
  
    private MemberDao memberDao; // = new MemberDao();  
  
    public MemberRegisterService(MemberDao memberDao) {  
        this.memberDao = memberDao;  
    }  
}
```

- memberdao 객체를 생성자를 통해서 '주입'받고 있다.
- 이러한 방법이 유지 보수 관점에서 대단히 중요하다.

객체 조립기

- 객체를 생성하기 위해서 의존을 주입한다고 배웠는데 그러면 실제 객체를 생성하는 곳은 어디일까??
- 보통 자바에서는 main()을 생성할 수 있는데 보통 객체를 생성하고 주입하는 클래스를 따로 작성한다.
- 이런 클래스를 서로 다른 객체를 주입한다고 해서 이런 클래스를 조립기라고 한다.
- 객체 조립기 클래스를 만들어 본다. --Assembler



객체 조립기

- 객체 조립기 클래스를 만들어 본다.

- spring.assembler

- Assembler.java

이부분이 에러가 난다면
changePwdSvc클래스도 생성자를
통해 주입받도록 수정하자

```
public class Assembler {  
    private MemberDao memberDao;  
    private MemberRegisterService regSvc;  
    private ChangePasswordService pwdSvc;  
  
    public Assembler() {  
        memberDao = new MemberDao();  
        regSvc = new MemberRegisterService(memberDao);  
        pwdSvc = new ChangePasswordService(memberDao);  
    }  
  
    public MemberDao getMemberDao() {  
        return memberDao;  
    }  
  
    public MemberRegisterService getMemberRegisterService() {  
        return regSvc;  
    }  
  
    public ChangePasswordService getChangePasswordService() {  
        return pwdSvc;  
    }  
}
```

객체 조립기

- 객체 조립기를 활용한 Main을 만들어서 동작시켜본다
 - spring.main
 - MainForAssembler.java

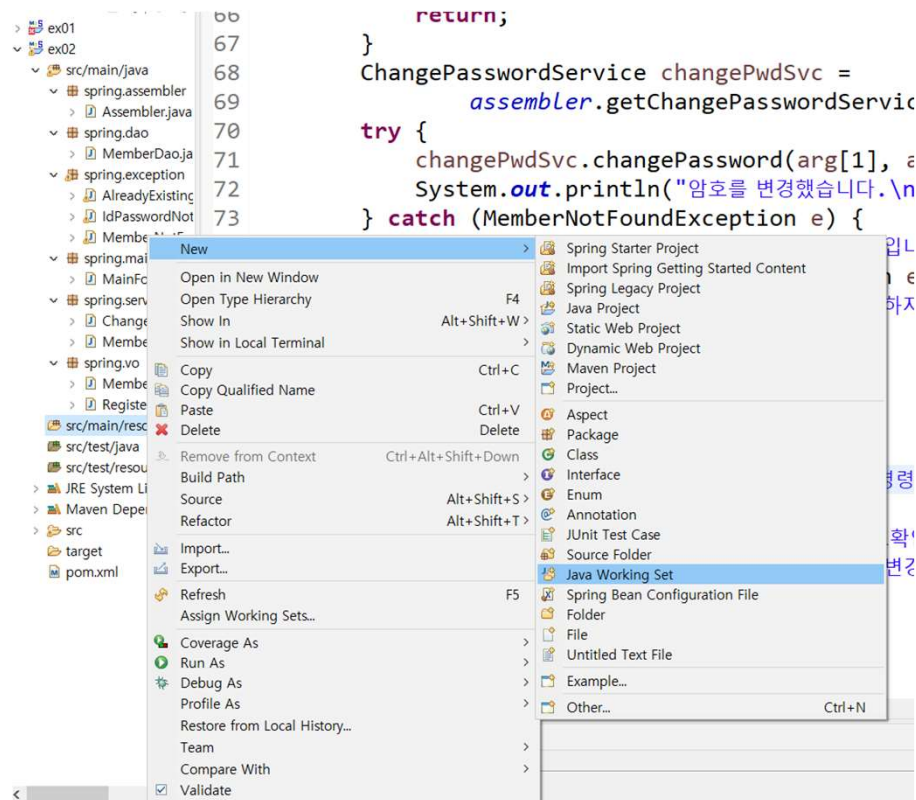
[MainForAssembler 클래스의 내용은 업로드된 자료를 참조]

스프링 DI설정

- 스프링은 앞서 본 조립기와 동일한 기능을 가진다.
- 적절히 필요한 객체를 생성하고 생성한 객체에 의존을 주입해 준다.
- 다만 앞서 본 Assembler와의 차이점은 스프링이 좀 더 범용적으로 사용 가능한 조립기라는 점이다.
- 우선 스프링이 어떤 객체를 생성하고 의존을 어떻게 주입하는지 설정 정보를 작성해 본다

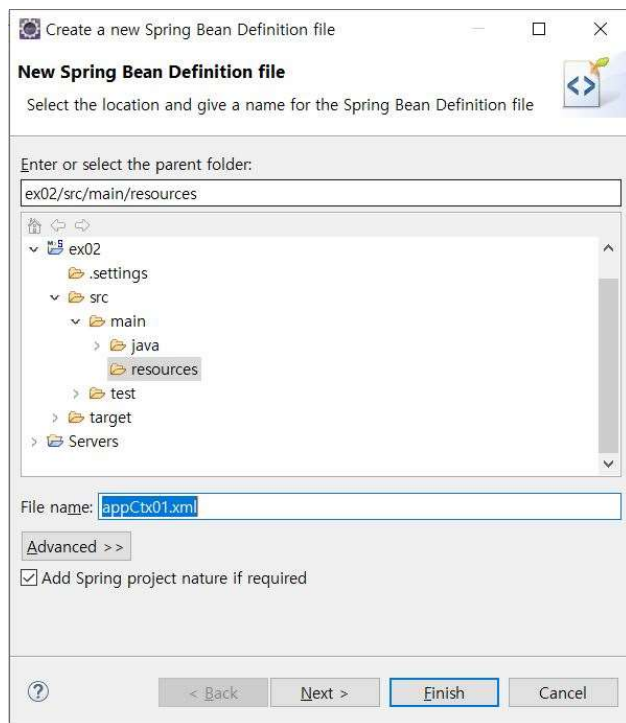
스프링 DI설정

Src/main/resources 에서 우클릭후 new => spring Bean configuration File로 생성



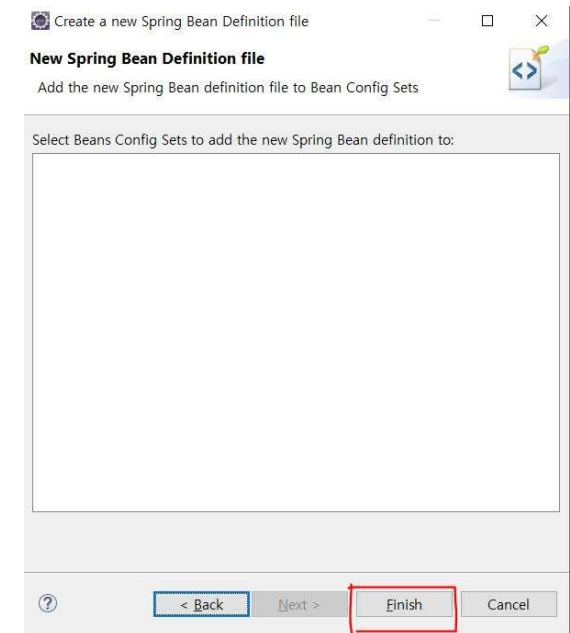
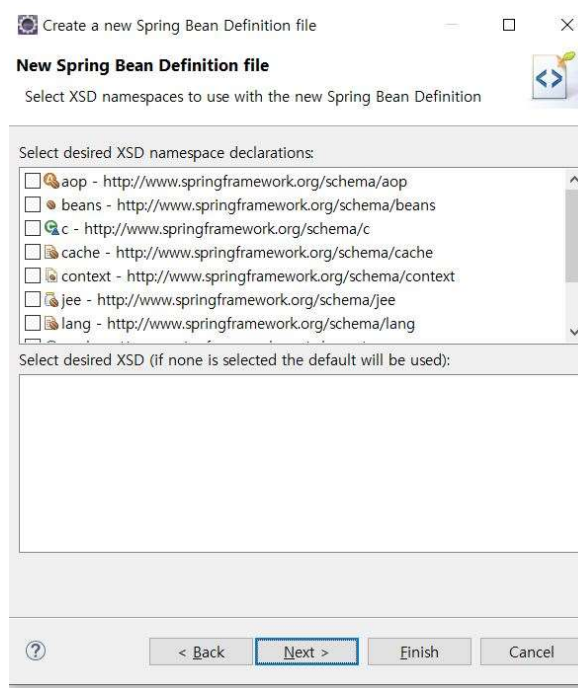
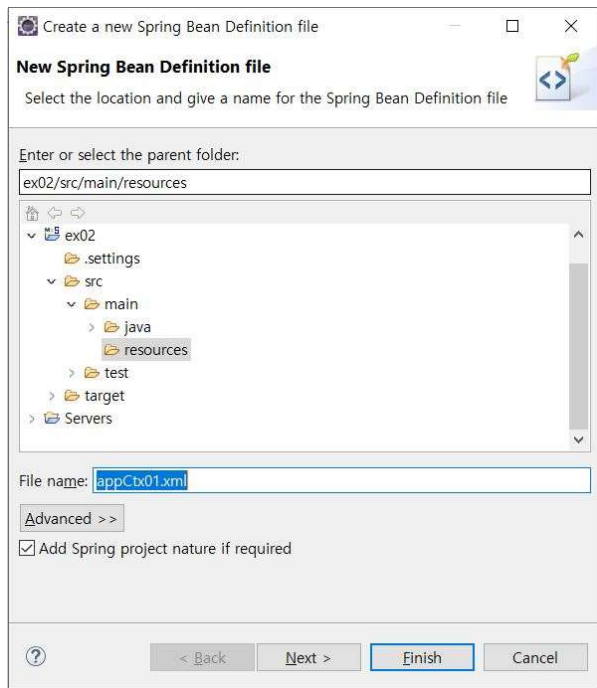
스프링 DI설정

파일명은 appCtx01.xml로 생성한다



스프링 DI설정

아무런 namespace를 사용하지 않을 것이므로 next -> next-> finish를 해서 파일을 생성한다



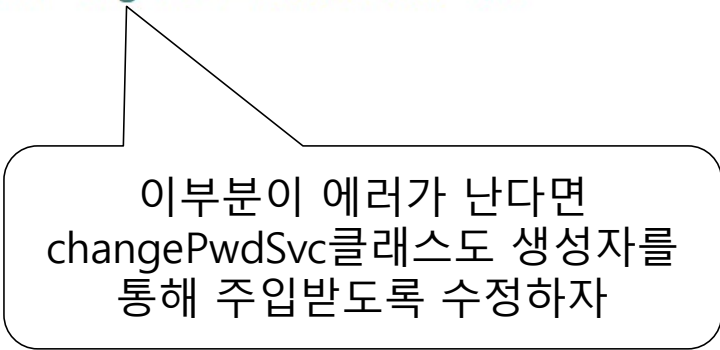
스프링 DI설정

스프링 설정파일에 빈(Beans) 객체를 생성한다.

```
<bean id="memberDao" class="spring.dao.MemberDao">  
</bean>
```

```
<bean id="memberRegSvc" class="spring.service.MemberRegisterService">  
  <constructor-arg ref="memberDao" />  
</bean>
```

```
<bean id="changePwdSvc" class="spring.service.ChangePasswordService">  
  <constructor-arg ref="memberDao" />  
</bean>
```



이부분이 에러가 난다면
changePwdSvc클래스도 생성자를
통해 주입받도록 수정하자

스프링 DI설정

- MainForSpring01을 만들어 본다. (예제4)
(기존의 MainForAssembler을 복사해서 만든다)

- 다만 Assemble 조립기를 사용하지 않고 스프링 설정파일을 사용해서 객체를 생성해야 하므로 그부분의 설정만 별도로 작성한다.

```
public class MainForSpring01 {  
  
    private static ApplicationContext ctx = null;  
  
    public static void main(String[] args) throws Exception {  
        ctx = new GenericXmlApplicationContext("classpath:appCtx01.xml");  
  
        ChangePasswordService changePwdSvc =  
            ctx.getBean("changePwdSvc", ChangePasswordService.class);  
    }  
}
```

스프링 DI설정

- 설정파일을 통해 스프링 컨테이너를 생성해야 한다.

```
ApplicationContext ctx = new GenericXmlApplicationContext("classpath:appCtx.xml");
```

- 생성 된 컨테이너로부터 객체를 가져다 사용할 수 있다

```
MemberRegisterService regSvc =  
    ctx.getBean("memberRegSvc", MemberRegisterService.class);
```

스프링 DI설정 – DI생성방식 : 생성자 방식

- 앞서 살펴 본 바는 생성자를 통해서 의존 객체를 주입 받았다.

```
public class MemberRegisterService {  
    private MemberDao memberDao;  
  
    public MemberRegisterService(MemberDao memberDao) {  
        this.memberDao = memberDao;  
    }  
}
```

- 스프링 XML 설정에서도 생성자를 이용해서 의존 객체를 주입받기 위해 사용한 태그는 `<constructor-arg ref="memberDao" />` 이다.
- 생성자를 통해 전달할 의존 객체가 2개 이상이면 `constructor-arg`태그를 순서대로 사용하면 된다.

스프링 DI설정 – DI생성방식 : 생성자 방식

- 생성자를 통해 주입 받을 객체가 두개 이상 인 경우의 예.

- spring. printer

- MemberPrinter.java

- MemberListPrinter.java

- appCtx02.xml

스프링 DI설정 – DI생성방식 : 생성자 방식

- 생성자를 통해 주입 받을 객체가 두개 이상 인 경우의 예.

-spring. printer

- MemberPrinter.java

```
public class MemberPrinter {  
    public void print(Member member) {  
        System.out.printf(  
            "회원 정보: 아이디=%d, 이메일=%s, 이름=%s, 등록일=%tF\n",  
            member.getId(), member.getEmail(),  
            member.getName(), member.getRegisterDate());  
    }  
}
```

스프링 DI설정 – DI생성방식 : 생성자 방식

- 생성자를 통해 주입 받을 객체가 두개 이상 인 경우의 예.

-spring. printer

- MemberListPrinter.java

```
public class MemberListPrinter {  
  
    private MemberDao memberDao;  
    private MemberPrinter printer;  
  
    public MemberListPrinter(MemberDao memberDao, MemberPrinter printer) {  
        this.memberDao = memberDao;  
        this.printer = printer;  
    }  
  
    public void printAll() {  
        Collection<Member> members = memberDao.selectAll();  
        for (Member m : members) {  
            printer.print(m);  
        }  
    }  
}
```

스프링 DI설정 – DI생성방식 : 생성자 방식

- 생성자를 통해 주입 받을 객체가 두개 이상 인 경우의 예.
- appCtx02.xml

(appCtx01에 등록된 빈 객체에 다음 내용을 추가해서 appCtx02를 만든다.

```
<bean id="listPrinter" class="spring.printer.MemberListPrinter">  
    <constructor-arg ref="memberDao" />  
    <constructor-arg ref="memberPrinter" />  
</bean>  
  
<bean id="memberPrinter" class="spring.printer.MemberPrinter">  
</bean>
```

스프링 DI설정 – DI생성방식 : 생성자 방식

- 새롭게 만든 appCtx02.xml를 사용하는 Main클래스를 만들어 보자

[MainForSpring02 클래스의 내용은 업로드된 자료를 참조]

스프링 DI설정 – DI생성방식 : 설정 메소드 방식

- 스프링은 생성자 뿐 아니라 setter 메소드를 통해서 의존 객체를 주입 받는 방식을 지원 한다.
- 프로퍼티 설정 메소드는 다음과 같다.
 - 메소드의 이름은 set으로 시작한다.
 - set 뒤에 붙는 이름은 프로퍼티 이름이며 첫글자는 대문자로 한다.
 - 한 개의 파라미터를 가지며, 파라미터 타입이 곧 프로퍼티의 타입이다.
- 스프링 설정 xml파일에서는
`<property name="memberDao" ref="memberDao" />`
형태로 객체를 주입받는다.

스프링 DI설정 – DI생성방식 : 설정 메소드 방식

- Setter메서드로 주입받는 예제를 알아보자

-spring. printer

- MemberInfoPrinter.java

```
public class MemberInfoPrinter {  
  
    private MemberDao memDao;  
    private MemberPrinter printer;  
  
    public void setMemberDao(MemberDao memberDao) {  
        this.memDao = memberDao;  
    }  
  
    public void setPrinter(MemberPrinter printer) {  
        this.printer = printer;  
    }  
  
    public void printMemberInfo(String email) {  
        Member member = memDao.selectByEmail(email);  
        if (member == null) {  
            System.out.println("데이터 없음\n");  
            return;  
        }  
        printer.print(member);  
        System.out.println();  
    }  
}
```

스프링 DI설정 – DI생성방식 : 설정 메소드 방식

- Setter메서드로 주입받는 예제를 알아보자
-appCtx02.xml에 다음 내용 추가

```
<bean id="infoPrinter" class="spring.printer.MemberInfoPrinter">  
    <property name="memberDao" ref="memberDao" />  
    <property name="printer" ref="memberPrinter" />  
</bean>
```

[MainForSpring02 클래스의 수정된 내용은 업로드된 자료를 참조 - MainForSpring03]

스프링 DI설정 – 기본 데이터 타입 값 설정

- 스프링 설정으로 다른 객체를 주입받기 위해서 <constructor-arg>태그나 <property> 태그의 ref 속성을 사용했다
- 그러나 객체를 주입받는 것이 아닌 기본 타입의 값을 주입받을 때는 어떻게 설정해야 할까?
 - 다음 예제를 살펴본다

스프링 DI설정 – 기본 데이터 타입 값 설정

- spring.printer
 - VersionPrinter.java

```
public class VersionPrinter {  
  
    private int majorVersion;  
    private int minorVersion;  
  
    public VersionPrinter() {  
    }  
  
    public VersionPrinter(int majorVersion, int minorVersion) {  
        this.majorVersion = majorVersion;  
        this.minorVersion = minorVersion;  
    }  
  
    public void setMajorVersion(int majorVersion) {  
        this.majorVersion = majorVersion;  
    }  
  
    public void setMinorVersion(int minorVersion) {  
        this.minorVersion = minorVersion;  
    }  
  
    public void print() {  
        System.out.printf("이 프로그램의 버전은 %d.%d입니다.\n\n",  
            majorVersion, minorVersion);  
    }  
}
```

스프링 DI설정 – 기본 데이터 타입 값 설정

- appCtx02.xml에 다음 내용을 추가해본다
- 생성자와 setter메서드를 사용하는 방법 두가지 모두 테스트 해보자

```
<!--  
<bean id="versionPrinter" class="spring.VersionPrinter">  
    <constructor-arg value="4" />  
    <constructor-arg value="1" />  
</bean>  
-->
```

생성자

```
<bean id="versionPrinter" class="spring.printer.VersionPrinter">  
    <property name="majorVersion" value="4" />  
    <property name="minorVersion">  
        <value>1</value>  
    </property>  
</bean>
```

Setter 메서드

스프링 DI설정 – 기본 데이터 타입 값 설정

[MainForSpring02 클래스의 수정된 내용은 업로드된 자료를 참조 - MainForSpring03]

스프링 두개의 설정 파일 사용하기

- 스프링 설정파일을 XML에 작성했다. 그러나 모든 설정을 하나의 XML에 작성하면 설정이 늘어남에 따라 관리하기가 어려워 질 것이다.
- 스프링은 이런 XML을 효율적으로 관리하기 위해서 두개 이상의 xml로 나눠서 관리 할 수 있는 기능을 제공한다.
- 나눠진 두개 이상의 XML을 사용하는 방법
 - 첫번째. 하나의 xml에 import 태그로 합치는 방법
 - 두번째. GenericXmlApplicationContext의 생성자 매개변수를 여러 개의 xml 주소를 담은 배열로 가져오는 방법

스프링 두개의 설정 파일 사용하기

첫번째. 하나의 xml에 import 태그로 합치는 방법

- appCtx02.xml을 Ctx01.xml과 Ctx02.xml 두개 적당히 분리한 다음 두개를 합쳐는 예제
- Ctx01.xml

```
<import resource="classpath:Ctx02.xml" />

<bean id="memberDao" class="spring.dao.MemberDao">
</bean>

<bean id="memberRegSvc" class="spring.service.MemberRegisterService">
    <constructor-arg ref="memberDao" />
</bean>

<bean id="changePwdSvc" class="spring.service.ChangePasswordService">
    <constructor-arg ref="memberDao" />
</bean>
```

스프링 두개의 설정 파일 사용하기

첫번째. 하나의 xml에 import 태그로 합치는 방법

- appCtx02.xml을 Ctx01.xml과 Ctx02.xml 두개 적당히 분리한 다음 두개를 합쳐는 예제
- Ctx02.xml

```
<bean id="listPrinter" class="spring.printer.MemberListPrinter">
    <constructor-arg ref="memberDao" />
    <constructor-arg ref="memberPrinter" />
</bean>

<bean id="memberPrinter" class="spring.printer.MemberPrinter">
</bean>

<bean id="infoPrinter" class="spring.printer.MemberInfoPrinter">
    <property name="memberDao" ref="memberDao" />
    <property name="printer" ref="memberPrinter" />
</bean>

<bean id="versionPrinter" class="spring.printer.VersionPrinter">
    <property name="majorVersion" value="4" />
    <property name="minorVersion">
        <value>1</value>
    </property>
</bean>
```

스프링 두개의 설정 파일 사용하기

첫번째. 하나의 xml에 import 태그로 합치는 방법

- appCtx02.xml을 Ctx01.xml과 Ctx02.xml 두개 적당히 분리한 다음 두개를 합쳐는 예제
[MainForSpring02 클래스의 수정된 내용은 업로드된 자료를 참조 - MainForSpring04]

스프링 두개의 설정 파일 사용하기

두번째. GenericXmlApplicationContext의 생성자 매개변수를 여러 개의 xml 주소를 담은 배열로 가져오는 방법

- appCtx02.xml을 Conf01.xml과 Conf02.xml 두개 적당히 분리한 다음 Main에서 합쳐보자
- Conf01.xml

```
<bean id="memberDao" class="spring.dao.MemberDao">
</bean>

<bean id="memberRegSvc" class="spring.service.MemberRegisterService">
    <constructor-arg ref="memberDao" />
</bean>

<bean id="changePwdSvc" class="spring.service.ChangePasswordService">
    <constructor-arg ref="memberDao" />
</bean>
```

스프링 두개의 설정 파일 사용하기

두번째. GenericXmlApplicationContext의 생성자 매개변수를 여러 개의 xml 주소를 담은 배열로 가져오는 방법

- appCtx02.xml을 Conf01.xml과 Conf02.xml 두개 적당히 분리한 다음 Main에서 합쳐보자
- Conf02.xml

```
<bean id="ListPrinter" class="spring.printer.MemberListPrinter">
    <constructor-arg ref="memberDao" />
    <constructor-arg ref="memberPrinter" />
</bean>

<bean id="memberPrinter" class="spring.printer.MemberPrinter">
</bean>

<bean id="infoPrinter" class="spring.printer.MemberInfoPrinter">
    <property name="memberDao" ref="memberDao" />
    <property name="printer" ref="memberPrinter" />
</bean>

<bean id="versionPrinter" class="spring.printer.VersionPrinter">
    <property name="majorVersion" value="4" />
    <property name="minorVersion">
        <value>1</value>
    </property>
</bean>
```

스프링 두개의 설정 파일 사용하기

두번째. GenericXmlApplicationContext의 생성자 매개변수를 여러 개의 xml 주소를 담은 배열로 가져오는 방법

- appCtx02.xml을 Conf01.xml과 Conf02.xml 두개 적당히 분리한 다음 Main에서 합쳐보자

[MainForSpring02 클래스의 수정된 내용은 업로드된 자료를 참조 - MainForSpring06]