어노테이션 기반 객체 생성

```
3 public class MemberService {
4⊖    public boolean update(String memberId)
5     {
6         System.out.println(memberId +
7                 "의 정보를 수정하였습니다.");
8         return true;
9     }
10 }
```

어노테이션 기반 객체의 생성

<bean /> ==> @Bean

어노테이션 기반 객체 생성

```
 6 @Configuration
 7 public class SpringConfig {
 8     @Bean
 9     public MemberService member1()
10     {
11         return new MemberService();
12     }
13 }
14
```

** 다음과 같은 xml 설정과 동일한 빈을 정의한다.

```
<bean id="member1" class="com.exam4.MemberService"/>
```

어노테이션 기반 객체 생성

```
12 public class HelloApp {
13
14⊖     public static void main(String[] args) {
15         // TODO Auto-generated method stub
16
17
18         ApplicationContext context
19         = new AnnotationConfigApplicationContext(SpringConfig.class);
20
21         MemberService m= context.getBean("member1", MemberService.class);
22         m.update("tiger");
23     }
24 }
```

어노테이션 기반 객체 생성시 의존 관계설정

```java
public class UpdateInfo {
    private String id;

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }
}
```

```java
public class MemberService {
    UpdateInfo info;
    public void setInfo(UpdateInfo info) {
        this.info = info;
    }
    public UpdateInfo getInfo() {
        return info;
    }
    public boolean update(String memberId)
    {
        System.out.println(memberId + "의 정보를 "+
                info.getId()+"로 수정하였습니다.");
        return true;
    }
}
```

어노테이션 기반 객체 생성시 의존 관계설정

```
 7 @Configuration
 8 public class SpringConfig {
 9     @Bean
10     public MemberService member1()
11     {
12         MemberService member = new MemberService();
13         member.setInfo(info1());
14         return member;
15     }
16
17     @Bean
18     public UpdateInfo info1()
19     {
20         UpdateInfo info = new UpdateInfo();
21         info.setId("lion");
22         return info;
23     }
24 }
25
```

@Bean 어노테이션의 autowire 속성을 이용한 연관 관계 자동 설정

```
 8 @Configuration
 9 public class SpringConfig {
10⊖    @Bean(autowire=Autowire.BY_NAME)
11     public MemberService member1()
12     {
13          MemberService member = new MemberService();
14          //member.setInfo(info1());
15          return member;
16     }
17
18⊖    @Bean
19     public UpdateInfo info()
20     {
21          UpdateInfo info = new UpdateInfo();
22          info.setId("lion");
23          return info;
24     }
25 }
```

** 다음과 같은 xml 설정과 동일한 빈을 정의한다.

```
<bean    id="member1" class="com.exam4.MemberService"
         autowire="byName"/>

<bean id="info" class="com.exam4.UpdateInfo">
    <property name="id" value="lion"/>
</bean>
```

한개 이상의 @Configuration 어노테이션 클래스의 사용

```
12 public class HelloApp {
13
14⊖    public static void main(String[] args) {
15         // TODO Auto-generated method stub
16
17         ApplicationContext context
18         = new AnnotationConfigApplicationContext(SpringConfig.class,
19                 ArticleServiceConfig.class);
20
21         MemberService m= context.getBean("member1", MemberService.class);
22         m.update("tiger");
23     }
24 }
```

xml 설정 파일에서 @Configuration 어노테이션 클래스 사용하기

```
<bean
   class="org.springframework.context.annotation.ConfigurationClassPostProcessor"/>

<bean class="com.exam4.SpringConfig"/>
```

or

```
<context:component-scan base-package="com.exam4"/>
```

@Configuration설정 클래스에서 xml사용하기 - @ImportResource사용

beans4.xml

```
n="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"

    <bean id="info" class="com.exam4.UpdateInfo">
        <property name="id" value="lion"/>
    </bean>

</beans>
```

SpringConfig.java

```
.exam4;

import org.springframework.beans.factory.annotation.Autowire;

@Configuration
@ImportResource("classpath:/beans4.xml")
public class SpringConfig {
    @Bean(autowire=Autowire.BY_NAME)
    public MemberService member1()
    {
        MemberService member = new MemberService();
        return member;
    }
}
```

두개 이상의 xml설정 파일의 사용

```
 9 @Configuration
10 @ImportResource({"classpath:/beans4.xml","classpath:/beans5.xml" })
11 public class SpringConfig {
12     @Bean(autowire=Autowire.BY_NAME)
13     public MemberService member1()
14     {
15         MemberService member = new MemberService();
16         return member;
17     }
18 }
19
```

서로 다른 @Configuration 어노테이션 클래스 간의 의존 관계 설정

```java
@Configuration
public class UpdateConfig {

    @Bean
    public UpdateInfo info()
    {
        UpdateInfo info = new UpdateInfo();
        info.setId("cat");
        return info;
    }
}
```

```java
@Configuration
public class SpringConfig {
    @Autowired
    private UpdateInfo info;

    @Bean
    public MemberService member1()
    {
        MemberService member = new MemberService();
        member.setInfo(info);
        return member;
    }
}
```

서로 다른 @Configuration 어노테이션 클래스 간의 의존 관계 설정

```
12 public class HelloApp {
13
14⊖    public static void main(String[] args) {
15         // TODO Auto-generated method stub
16
17         ApplicationContext context
18         = new AnnotationConfigApplicationContext(SpringConfig.class,
19                 UpdateConfig.class);
20
21         MemberService m= context.getBean("member1", MemberService.class);
22         m.update("tiger");
23     }
24 }
```

** 두 개의 파일을 모두 포함해야 함.
** 개발자가 모든 @Configuration 목록을 기억해야 하는 번거로움이 있음.

서로 다른 @Configuration 어노테이션 클래스 간의 의존 관계 설정

```
10 @Configuration
11 public class SpringConfig {
12     //@Autowired
13     //private UpdateInfo info;
14
15     //@Bean
16     @Bean(autowire=Autowire.BY_NAME)
17     public MemberService member1()
18     {
19         MemberService member = new MemberService();
20         //member.setInfo(info);
21         return member;
22     }
23 }
```

** autowire 속성을 이용하여 연관 관계 자동 설정 할 수 있음.

여러개의 @Configuration 클래스를 하나로 묶기 - @Import

```
11 @Configuration
12 @Import({UpdateConfig.class,ShopConfig.class})
13 public class SpringConfig {
14     //@Autowired
15     //private UpdateInfo info;
16
17     //@Bean
18     @Bean(autowire=Autowire.BY_NAME)
19     public MemberService member1()
20     {
21         MemberService member = new MemberService();
22         //member.setInfo(info);
23         return member;
24     }
25 }
```

```
12 public class HelloApp {
13     public static void main(String[] args) {
14
15         ApplicationContext context
16         = new AnnotationConfigApplicationContext(SpringConfig.class);
17
```

모든 @Configuration 목록을 기억할 필요 없이 @Import 어노테이션이 적용된 클래스만 기억하면 손쉽게 설정 정보 추적이 가능

라이프 사이클 처리

```
3  public class MemberService {
4      private UpdateInfo info;
5      public void init()
6      {
7          System.out.println("init 수행됨");
8      }
9
10     public void setInfo(UpdateInfo info) {
```

** initMethod, destoryMethod 속성을 이용해서 초기화, 종료 메서드를 지정할 수 있다.

```
11 @Configuration
12 @Import({UpdateConfig.class})
13 public class SpringConfig {
14     //@Autowired
15     //private UpdateInfo info;
16
17     //@Bean
18     @Bean(autowire=Autowire.BY_NAME, initMethod="init")
19     public MemberService member1()
20     {
21         MemberService member = new MemberService();
22         //member.setInfo(info);
23         return member;
24     }
25 }
```

객체의 유효 범위 설정- @Scope

```
12 @Configuration
13 @Import({UpdateConfig.class})
14 public class SpringConfig {
15     //@Autowired
16     //private UpdateInfo info;
17
18     //@Bean
19     @Bean(autowire=Autowire.BY_NAME, initMethod="init")
20     @Scope(value="prototype")
21     public MemberService member1()
22     {
23         MemberService member = new MemberService();
```

** 생략하면 디폴트는 singletone

```
12 public class HelloApp {
13     public static void main(String[] args) {
14
15         ApplicationContext context
16         = new AnnotationConfigApplicationContext(SpringConfig.class);
17
18         MemberService m1= context.getBean("member1", MemberService.class);
19         MemberService m2= context.getBean("member1", MemberService.class);
20         System.out.println("m1:"+m1);
21         System.out.println("m2:"+m2);
```