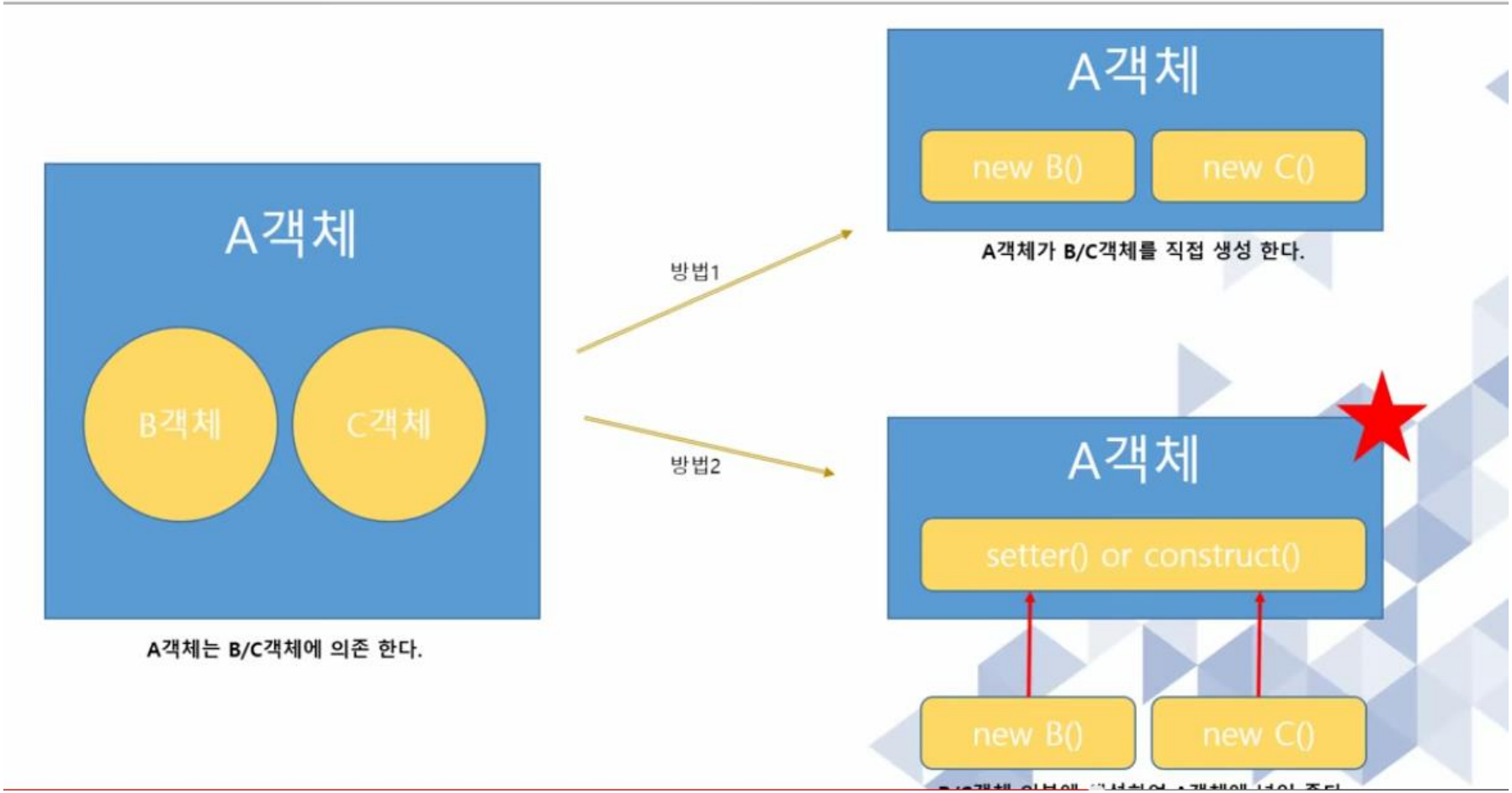
2. 의존성 주입

1. DI와 IOC 컨테이너
2. 빈생성

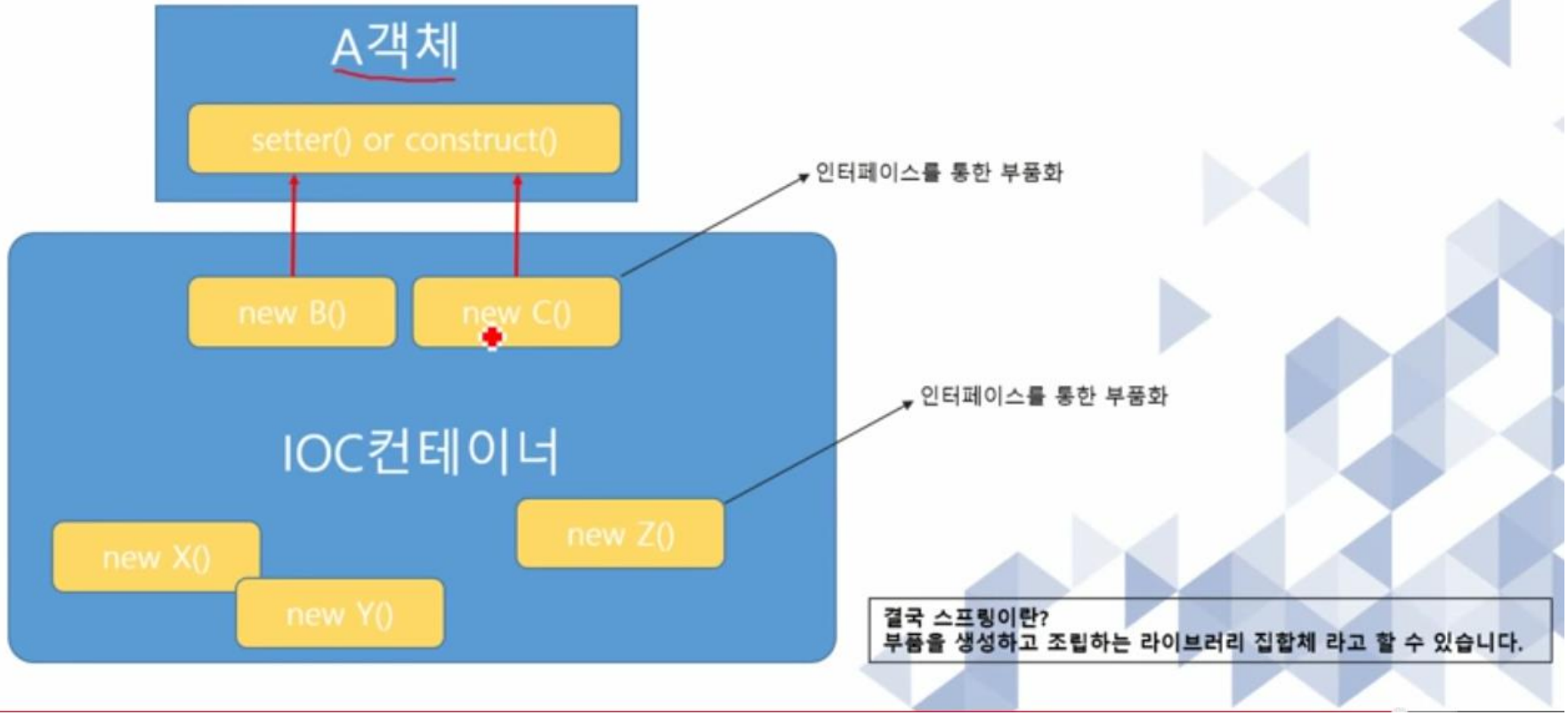
1. DI(Dependence Injection)와 IOC 컨테이너

- **객체간의 의존성**
  - 객체의 사용에 의해 발생
  - Interface에 의한 의존성 제거
  - factory에 의한 의존성 제거
  - Spring Framework 사용

1. DI(Dependence Injection)와 IOC 컨테이너



1. DI와 IOC 컨테이너



스프링을 사용하지 않은 프로젝트 (spring_3_1_ex1_springex)

```
public static void main(String[] args) {

    MyCalculator myCalculator = new MyCalculator();
    myCalculator.setCalculator(new Calculator());

    myCalculator.setFirstNum(10);
    myCalculator.setSecondNum(2);

    myCalculator.add();
    myCalculator.sub();
    myCalculator.mul();
    myCalculator.div();

}
```

스프링을 적용된 프로젝트 (spring_3_1_ex2_springex)

```
public static void main(String[] args) {

    String configLocation = "classpath:applicationCTX.xml";
    AbstractApplicationContext ctx = new GenericXmlApplicationContext(configLocation);
    MyCalculator myCalculator = ctx.getBean("myCalculator", MyCalculator.class);

    myCalculator.add();
    myCalculator.sub();
    myCalculator.mul();
    myCalculator.div();

}
```

실습

- **이전에 작성한 프로젝트 close**
- **새프로젝트 작성**

  프로젝트명: ExDI
- **새 패키지 작성(DI와 관련된 내용 패키지로 분류)**
  - 패키지명 : com.miya.section2.basi1
  - MessageBean 클래스 basic1에 작성

```
public class MessageBean{
        public void sayHello(String name){
                System.out.printl("How are You" + name + "님");
        }
}
```

App 클래스 수정

```
public class App{
        public static void main(String[] args){
                MessageBean bean=new MessageBean();
                bean.sayHello("홍길동");
        }
}
```

Interface에 의한 의존성 해결

- com.miya.section2.basic2 패키지 작성
- App클래스와 MessageBean클래스 복사
- IMessageBean 작성

```
public interface IMessageBean{
        public void sayHello(String name);
}
```

- MessageBean 수정

```
public class MessageBean implements IMessageBean{
        public void sayHello(String name){
                System.out.printl("How are You" + name + "님");
        }
}
```

App 수정

```
public class App{
        public static void main(String[] args){
                IMessageBean bean=new MessageBean();
                bean.sayHello("홍길동");
        }
}
```

MessageBeanKr 클래스 추가

```
public class MessageBeanKr implements IMessageBean{
        public void sayHello(String name){
                System.out.printl("안녕하세요 " + name + "님");
        }
}
```

- **App에서 MessageBeanKr 사용시 다음과 같이 수정**

```
public class App{
        public static void main(String[] args){
                IMessageBean bean=new MessageBeanKr();
                bean.sayHello("홍길동");
        }
}
```

- **Interface의 의존성 제거의 한계**

Factory Pattern에 의한 의존성 제거

- Factory Pattern 와 Singleton Pattern
- com.miya.section3.basic3 패키지 작성
- basic3에 basic2의 내용 복사
- factory : 인스턴스를 대신해 줌
- main은 factory의 주소만 알고 있으면 됨
- factory 클래스 작성 (클래스명: MessageFactory)

```
public class MessageFactory{
        public IMessage createMessage(){
                return new MessageBean;
        }
}
```

- **App 수정**

```
public class App{
        public static void main(String[] args){
                MessageFactory factory=new MessageFactory()
                IMessageBean bean=factory.createMessage()
                bean.sayHello("홍길동");
        }
}
```

- **Factory 가 여러 개 만들어지는 문제 발생**

Singleton Pattern

- **하나의 Factory만 생성**
  - 외부에서 인스턴스 생성 차단
    - :private 생성자 작성
  - 단 1개의 인스턴스만 생성
    - private static MessageFactory factory=new

```
public class MessageFactory{
        private MessageFactory(){}
        private staric MessageFactory factory=new MessageFactory();
        public static MessageFactory getInstance(){
                return factory;
        }
        public IMessage createMessage(){
                return new MessageBean;
        }
}
```

App수정

```
public class App{
        public static void main(String[] args){
                MessageFactory factory=MessageFactory.getInstance();
                IMessageBean bean=factory.createMessage()
                bean.sayHello("홍길동");
        }
}
```

Spring 사용

- **basic4 패키지 작성**
- **basic1의 내용으로 스프링 적용를 위해 복사**
- **Spring 다운로드**
  - spring.io사이트접속,-> project->SPRING FRAMEWORK ->Quick start 에서 버전을 3.2.X를 선택하고 다음을 pom.xml에 복사하고 pom.저장-자동 다운로드

```
<dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>3.2.12.RELEASE</version>
    </dependency
```

환경설정.xml 파일 작성

- 자동으로 factory 가지고 있음
- factory클래스를 가져올 설정파일 작성
- 프로젝트 루트에 설정파일 작성
- new->Spring Bean Configuration File 선택 or
- other->spring-> Spring Bean Configuration File 선택
- next -> 위치 프로젝트 루트-> file 명 : basic4.config.xml(임의로 작성)
- namespace 사용 상자에서 Beans 선택 -> beans 3.2버전 선택 후 finish 선택
- 설정파일에 다음을 추가한다.

```
<bean id="msgBean"
      class="com.miya.section3.basic4.Messagebean>
```

App 수정

```
public class App{
    public static void main(String[] args){
        BeanFactory factory =new XmlBeanFactory(new
                FileSystemResource("basic4_config.xml));
        MessageBean bean=factory.getBean("msgBean", MessageBean.class);
        bean.sayHello("홍길동");
    }
}
```

IMessageBean bean=factory,getInstance("msgBeab", IMessageBean.class);

IMessageBean을 작성하여 수정하여 보자

- **XmlBeanFactory**
  - Xml설정 파일에서 빈의 생성 및 설정, 관리 정보를 취득하기 위해 사용
- **Resource 인터페이스**
  - 여러가지 자원에 대한 추상적 접근을 위한 인터페이스
  - FileSystemResource
    - java.io.File을 다루기 위한 Resource 구현 클래스.
  - 기타 Resource 구현 클래스
    - ServletContextResource, UrlResource, ClassPathResource, InputStreamResource, ByteArrayResource, ResourceLoader 인터페이스 등

App 수정

```
public class App{
    public static void main(String[] args){
        BeanFactory factory =new DefaultListableBeanFactory();
        new XmlBeanDefinitionReader((BeanDefinitionRegistry)factory)
        .loadBeanDefinitions(new FileSystemResource("basic4_config.xml));
        IMessageBean bean=factory,getBean("msgBeab", IMessageBean.class);
        bean.sayHello("홍길동");
    }
}
```

- **DefaultListableBeanFactory**
  - 빈 팩토리를 구현하는 클래스
- **XmlBeanDefinitionReader**
  - xml빈을 정의하고 BeanDefinitionDocumentReader 인터페이스의 구현을 읽고 실제 XML 문서에 위임.
  - DefaultListableBeanFactory 또는 GenericApplicationContext에 적용
  - DOM 문서를로드하고, 문서에 BeanDefinitionDocumentReader을 적용, 주어진 빈 정의와 빈 팩토리를 BeanDefinitionRegistry 구현으로 문서에 등록.

환경 설정 파일 살펴보기

```
<bean id="msgBean"  name="msg"
      class="com.miya.section3.basic4.Messagebean>
```

- **id 속성 변수 역할**
- **name : 별명(alias), 여러 개 가능**
- **class : 사용한 클래스(패기지명.클래스)**

```
<bean id="msgBean"  name="msg, msg1, msg2, msg3"
      class="com.miya.section3.basic4.Messagebean>
```

여러 개의 name 속성 사용 예

```
public class App{
    public static void main(String[] args){
        BeanFactory factory =new DefaultListableBeanFactory();
        new XmlBeanDefinitionReader((BeanDefinitionRegistry)factory)
        .loadBeanDefinitions(new FileSystemResource("basic4_config.xml"));
        IMessageBean bean=factory.getBean("msg", IMessageBean.class);
        IMessageBean bean1=factory.getBean("msg1", IMessageBean.class);
        IMessageBean bean2=factory.getBean("msg2", IMessageBean.class);
        IMessageBean bean3=factory.getBean("msg3", IMessageBean.class);

        bean.sayHello("홍길동");
        bean1.sayHello("kim");
        bean2.sayHello("lee");
        bean3.sayHello("park");

    }
}
```

ApplicationContext 사용

```
public class App{
    public static void main(String[] args){
        ApplicationContext ctx=new
          FileSystemXmlApplicationContext("basic_config.xml");
        IMessageBean bean=ctx.getBean("msgBean", IMessageBean.class);
        bean.sayHello("홍길동");
    }
}
```

- **ApplicationContext 클래스**
  - BeanFactory를 확장한 것으로 BeanFactory의 기능 다양한 기능을 제공

환경설정 파일 경로 설정

```
ApplicationContext ctx=new FileSystemXmlApplicationContext(
                        "classpath:/com.miya/section3.basic4/basic_config.xml");
```

- **classpath:/패키지 경로/파일명**
  - com.miya.section3.basic4에 환경설정 파일이 있을 경우

    예:"classpath:/com.miya/section3/basic4/basic_config.xml"
- **경로 설정 패턴 문자 사용**
  - ** : 폴더를 나타냄
  - *:파일을 나타냄
  - 예:"classpath:/com.miya/section3/**/basic_config.xml"
  - 예:"classpath:/com.miya/section3/**/*.xml"

클래스 이용 환경설정

- **환경설정을 위한 클래스 작성**

Anotation이란?

```
@Configuration
public class ApplicationContextConfiguration{
        @Bean
        public IMessageBean  messageBean()
                return new MessageBean);{
        }
}
```

- **App 파일**

```
public class App{
    public static void main(String[] args){
        ApplicationContext  ctx=new
          AnnotationConfigApplicationContext(ApplicationContextConfiguration)
        IMessageBean bean=ctx,getBean("messageBean", IMessageBean.class);
        bean.sayHello("홍길동");
    }
}
```

@Bean의 name 속성

- 사용 예

  @Bean(name="msg")

  @Beaa(name={"msg","msg1","msg2"}

- App 클래스에서 사용 예

  IMessageBean bean=ctx,getBean("msg", IMessageBean.class);

@Bean의 init 및 destroy 속성

- init : 가장 먼저 부르는 함수
- destory : 마지막에 부르는 함수
- @Bean(init="initFnc")
- MessageBean 클래스에 InitFec() 메소드 작성

Bean의 의존성 주입

- **IoC 컨테이너**
  - 컨테이너에서 Spring 빈 인스턴스를 생성하고,생성된 빈인스턴스의 의존성을 관리
  - ApplicationContext 인터페이스는 IoC 컨테이너를 표현하고 스프링 빈 인스턴스를 생성, 설정하고 조합하는 기능 수행
  - IoC 컨터이너는 ApplicationContext 라고도 함

- **의존성 주입 방법**
  - 생성자를 이용한 방식
  - 필드를 사용하는 방식

생성자의 의한 의존성 주입

- 새 패키지 작성(com.miya.section3.di_exam)
- App 클래스, IMessageBean, MessageBean 작성
- IMessageBean, MessageBean 클래스 변경

```
public interface IMessageBean{
        public void sayHello();
}
```

```
public class MessageBeanKr implements IMessageBean{
    private String name;
    int age;
    Strimg greeting;
    public void sayHello(){
        System.out.printl(greeting +"!! " +  name + "님 나이가"
                + age +"입니다" );
    }
}
```

설정파일 작성

```
<bean id="msg"   class="com.miya.section3.di_exam.Messagebean>
```

App 수정

```
public class App{
      public static void main(String[] args){
            ApplicationContext   ctx=new
               FileSystemXmlApplicationContext(
                           "classpath:/com/miya/section2/di_exam.xml");
            IMessageBean bean=ctx,getBean("msg", IMessageBean.class);
            bean.sayHello();
      }
}
```

MessageBean 클래스에 생성자 추가

```
public class MessageBeanKr implements IMessageBean{
    private String name;
    int age;
    Strimg greeting;

    public MessageBean(String name, int age, String greeting){
        this.name=name;
        this.age=age;
        this.greeting=greeting;
    }
    public void sayHello(){
        System.out.printl(greeting +"!! " +  name + "님 나이가"
                + age +"입니다" );
    }
}
```

설정파일에서 생성자 값 주입

```
<bean id="msg"  class="com.miya.section3.di_exam.Messagebean>
        <constructor arg>
                <value>홍길동</value>
        </constructor>
        <constructor arg>
                <value>25</value>
        </constructor>
        <constructor arg>
                <value>안녕하세요</value>
        </constructor>

</bean>
```

setXXX()메소드에 값 전달

```
<bean id="msg"  class="com.miya.section3.di_exam.Messagebean>
        <constructor arg>
                <value>홍길동</value>
        </constructor>
        ……이하 생ㄹ략
        <property name="name" value="입꺽정"></property>
        또는
        <property name="name">
                <value>임꺽정</value>
        </property>


</bean>
```

index, type 속성 사용하기

```
<bean id="msg"  class="com.miya.section3.di_exam.Messagebean>
        <constructor arg index="0">
                <value>홍길동</value>
        </constructor>
        <constructor arg type="int">
                <value>25</value>
        </constructor>
        <constructor arg>
                <value>안녕하세요</value>
        </constructor>

</bean>
```

MessageBean에 setter 메소드 추카

```
public class MessageBeanKr implements IMessageBean{
    private String name;
    int age;
    Strimg greeting;

    public void setName(Strong name){
        this,name=name;
    }
    ..... 이하생략---
```