@Autowired

생성된 Bean을 자동으로 주입해주는 (자동으로 매핑해주는) 애너테이션

@Autowired 정의

```
@Target({ElementType.CONSTRUCTOR, ElementType.METHOD, ElementType.PARAMETER, ElementType.FIELD, ElementType.ANNOTATION_TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface Autowired {

    Declares whether the annotated dependency is required.
    Defaults to true.
    boolean required() default true;

}
```

`@Taget`을 보면 어느 식별자에 어노테이션을 적용하는지 알 수 있다.

@Autowired 를 적용할 수 있는 식별자

- 생성자
- 메서드
- 매개변수
- 필드
- 사용자 정의 애너테이션

@Autowired를 안쓴다면..

다음과 같은 상황이 있다고 가정해보자.

```
@Component
public class Dog{

    String dogName;

    public void setPetName(String petName) {
        dogName = petName;
    }

    public String getPetName() {
        return dogName;
    }
}
```

```
@Component
public class Person {

    public String name;
    public Dog pet;
```

```
    @Override
    public String toString() {
        return "Person{" +
                "name='" + name + '\'' +
                ", pet=" + pet.getClass().getSimpleName() +
                ", petName=" + pet.getPetName() +
                '}';
    }
}
```

모두 `@Component` 빈으로 등록하였다. 그러면 ApplicationContext의 getBean()을 통해 받아올 수 있겠지?

main() 에서 Person 빈의 pet으로 Dog 빈을 등록해보자.

```
@SpringBootApplication
public class AutowireApplication {

    public static void main(String[] args) {

        SpringApplication app = new
SpringApplication(AutowireApplication.class);
        app.setWebApplicationType(WebApplicationType.NONE);
        ApplicationContext ctx = app.run(args);

        // Person은 @Component이기 때문에 자동으로 빈으로 생성 되었다.
        Person hong = ctx.getBean(Person.class);
        hong.name = "Hong";

        // Dog, Cat, Animal도 @Component 이기 때문에 빈으로 생성되었다.
        // hong 에게 강아지를 선물하자
        Dog dog = ctx.getBean(Dog.class);
        dog.setPetName("Bingo");
        hong.pet = dog;

        System.out.println(hong);
    }
}
```

핵심은 Person의 pet 필드이다.

pet에는 무조건 null 대신 객체를 주입해야 하지 않나?

이때 pet에 `@Autowired`를 지정할 수 있다.

```java
@Component
public class Person {

    public String name;

    @Autowired // 이 부분!
    public Dog pet;

    // 중략...
}
```

이렇게 `@Autowired`로 매핑된 필드/생성자/메서드 등등은 `BeanFactory`에서 대상의 자료형에 일치하는 Bean 을 조회해 자동으로 넣어주는 기능을 제공한다.

그러면 main()의

```java
Dog dog = ctx.getBean(Dog.class);
dog.setPetName("Bingo");
hong.pet = dog;
```

이 작업을 다음과 같이 수정할 수 있다.

```java
hong.pet.setPetName("bingo");
```

Person 빈의 pet 필드는 자동으로 Dog 빈으로 wiring 되었다.

@Autowired 사용 시 주의사항

`@Autowired`는 참 편리한 기능을 제공하지만 다음의 경우 유의사항이 있다.

- wiring할 Bean이 없는 경우
- Wiring할 Bean이 여러 개인 경우

1. Wiring할 Bean이 없는 경우

> **[Bean 참고...]**
>
> - xml 설정파일에 Bean 등록이 없다거나..
> - `@Component`선언이 없다거나.. 혹은 선언은 되었는데 `@ComponentScan`이 수행되지 않았거나..
> - `@Configuration` 클래스의 `@Bean`으로 등록되지 않았거나..

위 경우 모두에 해당하면 Bean은 생성되지 않는다는 것을 기억하자!

Wiring 할 빈이 존재하지 않는다면 그를 가지고 있는 빈 또한 생성되지 않는다.

즉, Dog 클래스가 빈으로 등록되지 않았다면 Person 빈은 생성되지 않는다.

@Autowired에는 `required` 속성이 있다. 이 속성의 기본값이 `true`이기 때문에 해당 빈이 없으면 예외를 발생시킨 것이다. 다음과 같이 `false`로 변경해주면 Dog 빈이 없더라도 Person 빈은 생성될 수 있다.

```java
@Autowired(required="false") // 이 부분!
public Dog pet;
```

2. Wiring할 Bean이 여러 개인 경우

이번엔 Dog 외에 Cat 이라는 애완동물을 추가해보자.

그리고 이 둘은 Animal 인터페이스를 상속받게 했다.

```
@Component
public interface Animal {
    public void setPetName(String petName);
    public String getPetName();
}
```

```
@Component
public class Cat implements Animal{

    String catName;

    @Override
    public void setPetName(String petName) { catName = petName; }

    @Override
    public String getPetName() { return catName; }
}
```

```
@Component
public class Dog implements Animal{

    String dogName;

    @Override
    public void setPetName(String petName) {
        dogName = petName;
    }

    @Override
    public String getPetName() {
        return dogName;
    }
}
```

그리고 Person 에게도 Cat, Dog 이 두 타입을 모두 애완동물로 받을 수 있도록

pet 필드의 자료형을 Animal 로 바꾸었다.

```
@Component
public class Person {

    public String name;

    @Autowired
    public Animal pet;


    @Override
```

```
    public String toString() {
        return "Person{" +
                "name='" + name + '\'' +
                ", pet=" + pet.getClass().getSimpleName() +
                ", petName=" + pet.getPetName() +
                '}';
    }
}
```

그런데, 이렇게 된 경우 @Autowired 를 넣으니 문제가 발생하였다.

생성될 빈 중 2개의 빈이 모두 Animal 타입에 속하기 때문이다.

- Dog.class
- Cat.class

참고) Dog, Cat 빈은 생성되지만 Animal 빈은 생성되지않는다. (인터페이스라서)

해결방법은 다음과 같다.

- `@Primary`를 사용하여 Animal, Dog, Cat 중 최우선순위 클래스를 1개 지정한다.

```
@Component
@Primary
public class Dog implements Animal{ ... }
```

- `@Qualifer`를 사용하여 주입할 빈의 클래스 id를 지정해준다.

```
@Component
public class Person {
    ...
    @Autowired
    @Qualifier("cat")
    public Animal pet;
    ...
}
```

여기서 말하는 클래스 id는 클래스 이름에서 맨 앞글자만 소문자로 바꾼 String을 일컫는다.

@Qualifier("cat") --> Cat.class 빈

@Qualifier("dog") --> Dog.class 빈

@Qualifier("animal") --> Animal.class 빈

- 아예 List 형으로 모두 받는다.

```java
@Component
public class Person {

    public String name;

    @Autowired
    public List<Animal> pet;

    // toString() 은 에러나므로 삭제함.
}
```

```java
// main() 은 다음으로 수정
Person hong = ctx.getBean(Person.class);
hong.name = "Hong";
hong.pet.forEach(System.out::println);
```

결과