@Autowired annotation을 이용한 의존성 주입(Dependency Injection) 방법은 3가지가 있습니다.

1. 필드 주입 (Field Injection)
2. 수정자 주입 (Setter Injection)
3. 생성자 주입 (Constructor Injection)

3개의 방법 중 생성자 주입이 제일 권고되는 사항인데 왜 그런지는 각각의 주입방법에 대해서 살펴본 후에 정리해보겠습니다.

1. 필드 주입 (Field Injection)

필드 주입방식은 Class에 속한 Field 위에 @Autowired annotation을 붙여주시면 됩니다.
간단하게 Bean으로 등록할 클래스 2개를 만들어보겠습니다.

Developer Class

```
import org.springframework.stereotype.Component;

@Component
public class Developer {
}
```

WebProject Class

```
import org.springframework.stereotype.Component;

@Component
public class WebProject {
}
```

WebProject 에서 **Developer** 를 주입받을 때 필드 주입은 아래와 같이 사용할 수 있습니다.

WebProject Class

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

@Component
public class WebProject {

        @Autowired
        private Developer developer;
}
```

필드위에 @Autowired annotation이 붙으면 등록된 bean을 DI컨테이너에 의해서 자동으로 해당 필드에 주입해 줍니다.

2. 수정자 주입 (Setter Injection)

수정자 주입 코드를 작성해보겠습니다.

앞서 만들었던 **WebProject Class**에 아래와 같이 Setter method를 만들어 준 후에 @Autowired annotation을 붙여줍니다.

WebProject Class

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

@Component
public class WebProject {

	private Developer developer;

	@Autowired
	public void setDeveloper(Developer developer) {
		this.developer = developer;
	}
}
```

2.1 Lombok을 사용한 설정자 주입 (Setter Injection)

Lombok 버전은 1.18.12, JDK버전은 8입니다.

WebProject Class

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import lombok.Setter;

@Component
public class WebProject {

	@Setter(onMethod_= {@Autowired})
	private Developer developer;
}
```

DI Container가 bean으로 등록된 Developer객체를 setter메서드의 argument로 전달합니다.

3. 생성자 주입 (Constructor Injection)

생성자 주입 코드를 작성해보겠습니다.
생성자는 빈 생성자가 아닌 클래스의 필드를 파라미터로 사용하는 생성자여야 합니다.
앞서 만들었던 **WebProject Class**에 아래와 같이 생성자를 만들어 준 후에 @Autowired annotation을 붙여주세요.

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

@Component
public class WebProject {

	private Developer developer;

	@Autowired
	public WebProject(Developer developer) {
		this.developer = developer;
```

```
        }
}
```

3.1 Lombok을 사용한 생성자 주입(Constructor Injection)

Lombok 버전은 1.18.12, JDK버전은 8입니다.

WebProject Class

```
import org.springframework.stereotype.Component;
import lombok.AllArgsConstructor;

@AllArgsConstructor
@Component
public class WebProject {

        private Developer developer;

}
```

Lombok의 @AllArgsConstructor 어노테이션은 모든 필드를 파라미터로 받는 생성자를 만들어주는 역할을 합니다.
@Setter 어노테이션과 달리 @Autowired를 붙일 수 있는 속성은 없는데 Developer인스턴스는 정상적으로 생성자의 argument로 주입이 됩니다.
어째서일까요???
Spring 4.3 버전 이후부터 단일 생성자의 경우 묵시적 자동 주입이 가능합니다.
즉, 단일 생성자일 경우 @Autowired 어노테이션을 붙이지 않아도 자동으로 생성자 주입을 해주는 것입니다.

추가적으로 모든 필드를 파라미터로 받는 생성자가 아닌 특정 필드만 파라미터로 받는 생성자를 생성하고 싶을 때는 @RequiredArgsConstructor Class에 붙여주고, 파라미터로 받고 싶은 필드에는 @NonNull어노테이션 혹은 final을 붙여주시면 됩니다.

```
import org.springframework.stereotype.Component;
import lombok.NonNull;
import lombok.RequiredArgsConstructor;

@RequiredArgsConstructor
@Component
public class WebProject {

        private final Developer back_end_developer;

        @NonNull
        private Developer front_end_developer;

}
```

4. 왜 생성자 주입을 권고하는가??

@Autowired를 사용해서 의존성을 주입하는 방법은 앞에서 본 것과 같이 3가지의 방법이 있습니다.
그중에서 생성자 주입을 가장 권고하는 이유가 무엇인지 알아보겠습니다.

4.1 SRP(단일 책임의 원칙)를 위반할 확률이 줄어든다.

우리가 작성한 비즈니스 로직을 담당하는 클래스는 하나의 책임에 집중되어 있어야 합니다. 이는 객체지향 프로그래밍의 원칙 중 하나입니다.
생성자 주입을 사용하지 않고 필드 주입을 사용하게 될 경우 클래스 내부에 선언된 필드에 그저 @Autowired를 붙이는 것으로 쉽게 의존성을 주입해 사용할 수 있습니다.
쉽게 의존성을 주입할 수 있다는 것은 하나의 클래스가 여러 가지 기능을 담당하게 만들기도 쉽다는 얘기랑 같습니다.
그러나 생성자 주입을 사용하게 되면 생성자 파라미터에 사용하고자 하는 필드를 모두 넣어주어야 하기 때문에 코드가 길어지고 그로 인해 경각심을 가질 수 있습니다.

4.2 필드에 final을 선언할 수 있다.

생성자 주입을 제외한 필드, 수정자 주입은 final을 선언할 수 없습니다.
필드에 final을 붙이기 위해서는 클래스의 인스턴스가 생성될 때 final이 붙은 필드를 반드시 초기화해야 합니다.
Field/Setter Injection은 우선 인스턴스가 생성된 후에 해당 필드에 의존성 주입이 진행되므로 final을 붙일 수가 없습니다.
그러나 생성자 주입은 필드를 파라미터로 받는 생성자를 통해 클래스의 인스턴스가 생성될 때 의존성 주입이 일어나고, 이때 final이 붙은 필드가 초기화가 됩니다.
우리가 웹 개발을 할 때 Bean객체의 필드 값이 바뀌는 일은 거의 없을 겁니다. 그러므로 필드에 final을 붙여 불변성을 가지도록 하는 것이 좋습니다.

4.3 DI 컨테이너에 독립적인 테스트 코드를 작성할 수 있다.

개발을 할 때 테스트 코드를 작성하는 것은 매우 중요합니다.
필드 주입을 사용하게 되면 테스트 코드에서 어떻게 내부 필드에 인스턴스를 넣어 줄 수 있을까요???
아래와 같이 일반적인 JUnit을 사용하는 테스트 코드에서는 불가능합니다.

```
import org.junit.Test;
import com.advertising.star.autowiredtest.Developer;
import com.advertising.star.autowiredtest.WebProject;

public class TestWithoutDIContainer {

        @Test
        public void test() {
                Developer dev = new Developer();
                WebProject web =  new WebProject();
                //접근제한자가 public일 때만 이렇게 가능
                //web.developer = dev;
        }
}
```

왜냐하면 일반적으로 필드의 접근 제한자를 public으로 하게 되면 외부에서 필드의 값을 변경할 수 있으므로 대부분은 이를 방지하기 위해 private로 선언합니다.
Field Injection일 때 테스트 코드를 작성하기 위해서는 아래와 같이 DI컨테이너를 사용하는 테스트 코드를 작성해야 합니다.

```
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import com.advertising.star.autowiredtest.Developer;
import com.advertising.star.autowiredtest.WebProject;
import com.advertising.star.config.RootConfig;


@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes = RootConfig.class)
public class TestWithDIContainer {

        //Di컨테이너에 의해 내부 Developer필드에 의존성 주입가능
        @Autowired
        WebProject webProject;

        @Test
        public void test() {
           System.out.println(webProject);
        }
}
```

그러나 Constructor / Setter Injection을 사용하게 되면 아래와 같이 DI컨테이너에 독립적으로 테스트 코드를 작성할 수 있습니다.

```
import org.junit.Test;
import com.advertising.star.autowiredtest.Developer;
import com.advertising.star.autowiredtest.WebProject;

public class TestWithoutDIContainer {

        @Test
        public void test() {
                Developer dev = new Developer();

                //Setter Injection 사용할 때
                WebProject web =  new WebProject();
                web.setDeveloper(dev);

                //Constructor Injection 사용할 때
                WebProject web =  new WebProject(dev);

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

4.4 순환 참조를 발견할 수 있다.

순환 참조는 2개의 클래스가 서로를 참조하고 있는 상태로 마치 break point가 없는 재귀 함수가 실행될 때의 모습과 비슷합니다.
앞서 작성한 클래스를 아래와 같이 수정해 보겠습니다.

Developer Class

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import lombok.Setter;
import lombok.ToString;
```

```
@ToString
@Component
public class Developer {

//	Field Injection 사용 시
	@Autowired
	private WebProject webProject;

//	Setter Injection 사용 시
//	@Setter(onMethod_= {@Autowired})
//	private WebProject webProject;

	public void print() {
		System.out.println("developer print");
		webProject.print();
	}
}
```

WebProject Class

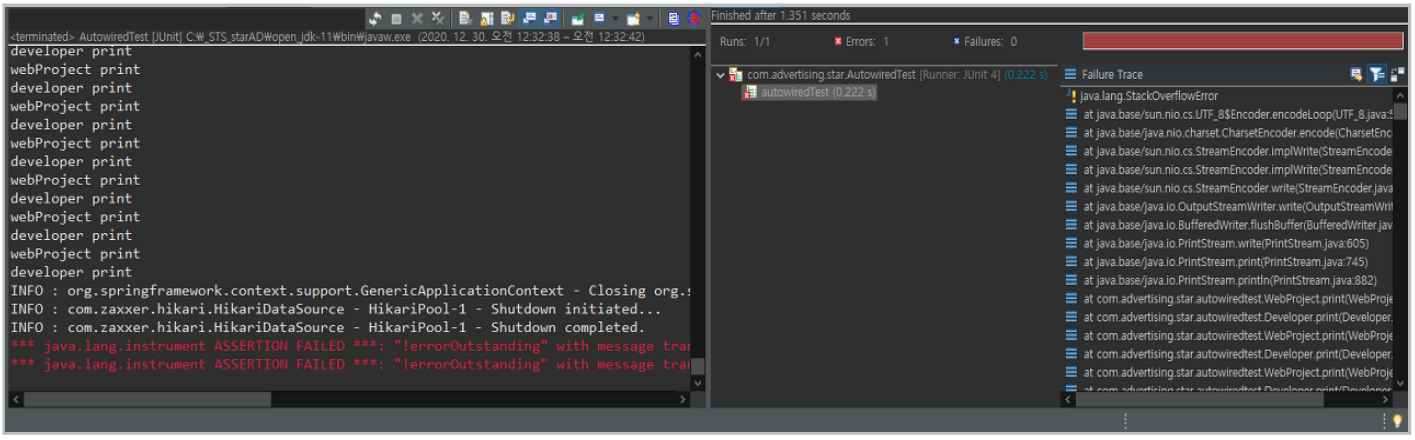
```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import lombok.Setter;
import lombok.ToString;

@ToString
@Component
public class WebProject {

//	Field Injection 사용 시
	@Autowired
	private Developer developer;

//	Setter Injeciton 사용 시
//	@Setter(onMethod_= {@Autowired})
//	private Developer developer;

	public void print() {
		System.out.println("webProject print");
		developer.print();
	}
}
```

위와 같이 2개의 클래스가 서로 순환 참조하는 형태일 때 Field/Setter Injection 두 가지의 경우는 서버를 실행해도 에러가 발생하지 않습니다.
둘 중 하나의 클래스에서 print() 메서드를 호출하였을 경우에 서로의 print() 메서드를 호출해가면서 stackoverflow 에러가 발생합니다.
웹서비스를 하고 있는 상황이라면 서버가 갑자기 뻗어버리게 되므로 좋지 않은 상황이 될 것입니다.

Test Class

```
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import com.advertising.star.autowiredtest.Developer;
import com.advertising.star.autowiredtest.WebProject;
import com.advertising.star.config.RootConfig;

@RunWith(SpringJUnit4ClassRunner.class)
```

```
@ContextConfiguration(classes = RootConfig.class)
public class CircularReferenceTest  {

	@Autowired
	WebProject webProject;

	@Autowired
	Developer developer;

	@Test
	public void circularReferenceTest() {
		webProject.print();
		developer.print();
	}
}
```



circular reference error log (field / setter Injection)

그러나 Constructor Injection을 사용하게 되면 bean이 생성되는 시점에 순환 참조를 발견할 수 있습니다.
2개의 클래스를 수정 후에 테스트해보겠습니다.

Developer Class

```
import org.springframework.stereotype.Component;
import lombok.RequiredArgsConstructor;
import lombok.ToString;

@RequiredArgsConstructor
@ToString
@Component
public class Developer {

	private final WebProject webProject;

	public void print() {
		System.out.println("developer print");
		webProject.print();
	}
}
```

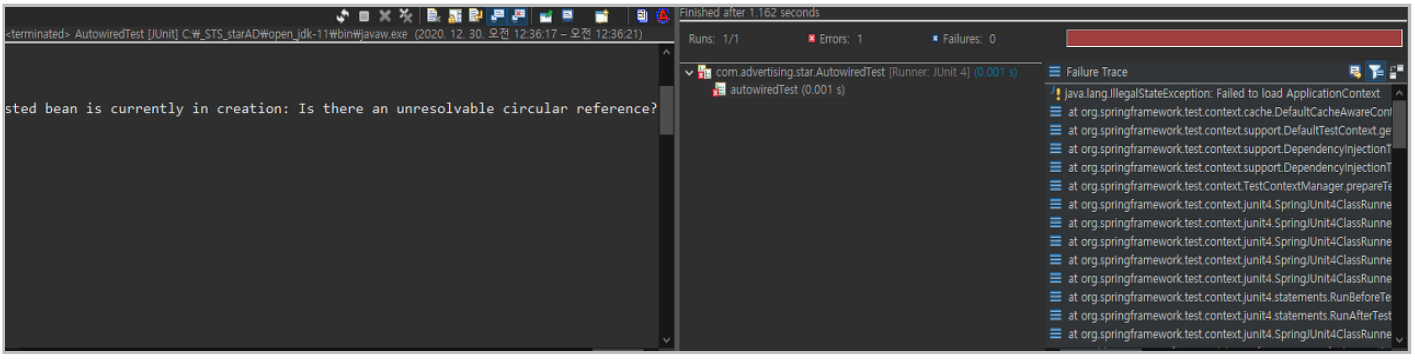
WebProject Class

```
import org.springframework.stereotype.Component;
import lombok.RequiredArgsConstructor;
import lombok.ToString;
```

```
@RequiredArgsConstructor
@ToString
@Component
public class WebProject {

	private final Developer developer;

	public void print() {
		System.out.println("webProject print");
		developer.print();
	}
}
```

Test Class

```
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import com.advertising.star.autowiredtest.Developer;
import com.advertising.star.autowiredtest.WebProject;
import com.advertising.star.config.RootConfig;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes = RootConfig.class)
public class CircularReferenceTest  {

	@Autowired
	WebProject webProject;

	@Autowired
	Developer developer;

	@Test
	public void circularReferenceTest() {
		webProject.print();
		developer.print();
	}
}
```



circular reference error log 일부 (constructor injeciton)

로그를 보면 circular reference로 인해 서버가 실행되지 않으며 BeanCurrentlyInCreationException이 발생하는 걸 볼 수 있습니다.
그러면 왜 생성자 주입일 때는 애플리케이션이 실행될 때 확인이 가능하고 필드나 수정자 주입일 때는 순환 참조가 확인이 안 되는 것일까요??
간단합니다. 생성자 주입의 경우 bean을 생성하는 시점에 의존성 주입이 돼야 하는데, 서로가 서로를 주입받아야 하는 상태이기 때문에 에러가 발생하는 것입니다.

필드/수정자 주입의 경우는 우선 bean이 생성된 후에 Field 혹은 Setter메서드에 의존성을 주입하므로 정상적으로 의존성이 주입되지만 비즈니스 로직을 수행하는 과정에서 에러가 발생하게 됩니다.

이 외에도 Constructor Injection을 권고하는 이유들이 더 있을 거라고 생각되지만, 지금까지 위에서 설명한 내용들만으로도 생성자 주입을 사용하는 것이 장점이 더 많다는 것을 알 수 있습니다.
그러므로 @Autowired를 사용할 때는 Constructor Injection을 사용하시는 것을 권장드립니다.