02_의존성 주입

스프링 프레임워크에서 의존성 주입은 ***ApplicatoinContext**를 통해서 이루어진다.

> 💡 ApplicatoinContext
>
> - 객체(Bean)와 객체의 의존성을 생성하고 관리하는 역할을 한다.

어플리케이션 클래스를 검색하여 특정 어노테이션(@Service, @Repository, @Controller 등)이 있는 클래스를 스프링 빈으로 등록한다.

의존성 주입 방법을 여러 가지가 있으며, 일반 적인 방법은 다음과 같다.

1. 생성자 주입

생성자를 통해 의존성을 주입한다.

객체가 생성될 때 필요한 모든 의존성을 반드시 이용하게 하므로 **가장 권장되는 방식**이다.

```
@Service
public class UserService{
    private final UserRepository userRepository;
```

```
    private final PasswordEncoder passwordEncoder;

    public UserService(UserRepository userRepository, PasswordEncoder p
asswordEncoder) {
        this.userRepository = userRepository;
        this.passwordEncoder = passwordEncoder;
    }
}
```

위 코드처럼 생성자가 한 개라면 Spring에서 알아서 주입을 해주지만, 생성자가 여러 개라면 **@Autowired 어노테이션**을 이용해서 의존성을 주입 받을 생성자를 정의 해야 한다.

```
@Service
public class UserService{
    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;

        // 다른 생성자
        public UserService(){
                ...초기화 코드
        }

        // 주입 받고자 하는 생성자
        @Autowired
    public UserService(UserRepository userRepository, PasswordEncoder p
asswordEncoder) {
        this.userRepository = userRepository;
        this.passwordEncoder = passwordEncoder;
    }
}
```

필드가 많거나, 필드의 정보가 추가/수정/삭제가 되게 된다면 생성자를 수정해주어야 하기 때문에 생성자를 직접 사용하지 않고, **주로 어노테이션을 사용**하며, 여러 기능을 제공해주는 **Lombok 라이브러리의 어노테이션 위주로 설명**

1. Spring 자체 어노테이션

@Autowired

```
@Service
public class UserService{
        @Autowired
    private final UserRepository userRepository;
}
```

1️⃣ 의존성 주입에서 사용하는 가장 대표적인 어노테이션

2️⃣ ❗ **필드 주입 방식**

3️⃣ private인데도 주입은 가능 → 리플렉션으로 동작

4️⃣ 명시적인 생성자가 없기 때문에 디버깅 시 **구조 파악이 어려움**

5️⃣ 실수로 **다른 값 재할당이 가능** → **위험성 증가**

2. Lombok 어노테이션

1. @RequiredArgsConstructor

```
@Service
@RequiredArgsConstructor
public class UserService{
    private final UserRepository userRepository;

    private String name; // 생성자에 포함 안됨
}
```

1️⃣ Lombok 어노테이션

2️⃣ **final 필드만을 대상**으로 생성자 자동 생성

3️⃣ 스프링은 **생성자가 하나만 있을 경우 자동으로 주입**

4️⃣ 생성자 주입은 테스트 용이, **불변성 보장**

5️⃣ ❗ **실무에서 가장 권장되는 방식**

💡 ❓ 실무에서 왜 선호하는지

- 필요한 의존성만 final로 선언하고 주입하기 때문에 여러 장점이 있다.

1. **불변성 보장**
2. **불필요한 의존성 주입 방지**
3. **가독성 향상**

2. @AllArgsConstructor

```
@Service
@AllArgsConstructor
public class UserService{
    private final UserRepository userRepository;

    private String name; // 생성자에 포함
}
```

1️⃣ **모든 필드**에 대해 생성자 생성

2️⃣ 테스트용 DTO나 임시 객체에는 좋지만, **의존성 주입에 사용하면 위험할 수 있다** (불변성, 의도 명확하지 않음)

3. @NoArgsConstructor

```
@NoArgsConstructor
public class User{
    private String name;
}
```

1️⃣ 기본 생성자를 만들어준다.

2️⃣ JPA 엔티티에는 필수 (@Entity는 기본 생성자 필요)

3️⃣ 단독 사용보다는 ORM과 함께 사용하는 경우가 많음

2. Setter 주입

setter 주입은 선택적 의존성이 있거나 런타임(프로그램 실행 중)에 의존성을 수정하려는 경우에 유용하다.

```
@Service
public class UserService{
    private final UserRepository userRepository;

    @Autowired
    public void setUserRepository(UserRepository userRepository){
            this.userRepository = userRepository;
    }
}
```

3. 필드 주입

의존성을 필드에 직접 주입하는 방법이다.

필드 주입의 장점은 단순성이지만, 몇 가지 단점이 있다.

의존성을 이용할 수 없는 런타임 오류가 발생할 수 있고, 테스트를 위해 의존성을 모의 객체로 만들 수 없기 때문에 클래스를 테스트하기가 더 어렵다.

```
@Service
public class UserService{
        @Autowired
    private UserRepository userRepository;
}
```