Chap03. Dependency Injection

1. Dependency Injection

1-1. Dependency Injection이란?

1-1-1. Dependency Injection

> 💡 Dependency Injection(의존성 주입, 이하 DI)은 객체 간의 `의존 관계`를 빈 설정 정보를 바탕으로 컨테이너가 자동적으로 연결해주는 것을 말한다. 이를 통해 객체 간의 `결합도`를 낮출 수 있으며 이로 인해 `유지보수성`과 `유연성`이 증가한다.

1-1-2. 의존 관계와 결합도

```
public class A {

  private B b = new B();

}

public class B {

}
```

`class A` 가 `class B` 를 필드로 가질 때 `A`는 `B`에 의존하게 된다.

```
public class A {

  /* 컴파일 에러 발생 */
  private B b = new B();

}

/* 클래스명이 B에서 NewB로 변경 */
public class NewB {

}
```

의존성이 강하다는 것은 한 객체가 변경되면 이에 의존하는 다른 객체들도 함께 변경되어야 한다는 것을 의미한다. `B`가 `NewB` 로 변경되면 해당 클래스를 필드로 가지고 있는 `A` 도 변경되어야 할 것이다. 이처럼 객체 간의 의존 관계가 강하게 묶여있을 때 `결합도` 가 높다고 표현한다. 이로 인해 유지보수성과 유연성이 저하될 수 있다.

```
public class A {

  /* 상위 타입을 필드로 설정 */
  private B b;

  /* 직접 객체를 생성하지 않고 생성자를 통해 전달 받음 */
  public A(B b) {
    this.b = b;
  }

}

/* 상위 타입으로 사용할 인터페이스 */
public interface B {

}

/* 인터페이스의 구현 클래스 */
public class NewB implements B {

}
```

이전의 코드와 비교하면 `NewB` 라는 구체적인 구현체의 타입을 사용하는 대신 `B` 라는 상위 인터페이스 타입으로 필드를 선언했다. 또한 직접 객체를 생성하는 구문도 없어졌고 생성자를 통해 전달 받도록 했다. 이렇게 변경하면 실제로 사용하는 구현체가 `NewB` 에서 또 다른 타입으로 변경 되더라도 `A` 의 코드는 변경 될 필요가 없다. 의존 관계가 느슨해지고 결합도가 낮아졌다고 할 수 있다.

2. DI 방법 알아보기

아래 코드는 테스트에 공통적으로 사용 할 `Account`, `PersonalAccount`, `MemberDTO` 클래스이다.

1. Account

```
public interface Account {

    /* 잔액 조회 */
    String getBalance();

    /* 입금 */
    String deposit(int money);

    /* 출금 */
    String withDraw(int money);

}
```

2. PersonalAccount

```
@RequiredArgsConstructor
public class PersonalAccount implements Account {
```

```
    private final int bankCode;       //은행코드
    private final String accNo;       //계좌번호
    private int balance;              //잔액

    @Override
    public String getBalance() {

        return this.accNo + " 계좌의 현재 잔액은 " + this.balance + "원 입니다.";
    }

    @Override
    public String deposit(int money) {

        String str = "";

        if(money >= 0) {
            this.balance += money;
            str = money + "원이 입금되었습니다.";
        } else {
            str = "금액을 잘못 입력하셨습니다.";
        }

        return str;
    }

    @Override
    public String withDraw(int money) {

        String str = "";

        if(this.balance >= money) {
            this.balance -= money;
            str = money + "원이 출금되었습니다.";
        } else {
            str = "잔액이 부족합니다. 잔액을 확인해주세요.";
        }

        return str;
    }

}
```

3. MemberDTO

```
@Getter @Setter @ToString
@NoArgsConstructor
@AllArgsConstructor
public class MemberDTO {

    private int sequence;                //회원번호
    private String name;                 //이름
    private String phone;                //휴대폰번호
    private String email;                //이메일
    private Account personalAccount;     //개인계좌

}
```

⇒ `Account(계좌)` 인터페이스를 구현한 `PersonalAccount(개인계좌)` 클래스가 있고 `MemberDTO` 는 `Account` 타입을 필드로 가지고 있다. (`MemberDTO` 는 `Account` 타입에 의존한다.)

2-1. XML Configuration

2-1-1. 생성자(Constructor) 주입

```
<bean id="account" class="com.ohgiraffers.common.PersonalAccount">
    <constructor-arg index="0" value="20"/>
    <constructor-arg index="1" value="110-234-567890"/>
</bean>

<bean id="member" class="com.ohgiraffers.common.MemberDTO">
    <constructor-arg name="sequence" value="1"/>
    <constructor-arg name="name" value="홍길동"/>
    <constructor-arg name="phone" value="010-1234-5678"/>
    <constructor-arg name="email" value="hong123@gmail.com"/>
    <constructor-arg name="personalAccount">
        <ref bean="account"/>
    </constructor-arg>
</bean>
```

bean 태그의 클래스 속성은 인터페이스 타입이 아닌 구현 클래스 타입으로 작성해야 한다. 따라서 `account` 빈 등록 시 class 속성에는 `Account` 인터페이스가 아닌 `PersonalAccount` 클래스를 사용한다.

`MemberDTO` 는 `Account` 타입을 의존하고 있기 때문에 `member` 빈 등록 시 `account` 빈을 참조하도록 `<constructor-arg>` 태그의 `ref` 속성을 작성한다. 생성자를 통해 의존성 객체를 전달하여 의존성을 주입하고 있으므로 이를 `생성자 주입` 이라 한다.

```
/* XML 설정 파일을 기반으로 ApplicationContext 객체 생성 */
ApplicationContext context
  = new GenericXmlApplicationContext("section01/xmlconfig/spring-context.xml");

/* MemberDTO 타입의 빈 가져오기 */
MemberDTO member = context.getBean(MemberDTO.class);

/* MemberDTO의 PersonalAccount 객체 출력 */
System.out.println(member.getPersonalAccount());
/* 10000원 입금 */
System.out.println(member.getPersonalAccount().deposit(10000));
/* 잔액 출력 */
System.out.println(member.getPersonalAccount().getBalance());
/* 5000원 출금 */
System.out.println(member.getPersonalAccount().withDraw(5000));
/* 잔액 출력 */
System.out.println(member.getPersonalAccount().getBalance());
```

▼ 실행 결과

```
PersonalAccount(bankCode=20, accNo=110-234-567890,
  balance=110-234-567890 계좌의 현재 잔액은 0원 입니다.)
110-234-567890 계좌의 현재 잔액은 0원 입니다.
10000원이 입금되었습니다.
110-234-567890 계좌의 현재 잔액은 10000원 입니다.
5000원이 출금되었습니다.
110-234-567890 계좌의 현재 잔액은 5000원 입니다.
```

2-1-2. 세터(Setter) 주입

```
<bean id="account" class="com.ohgiraffers.common.PersonalAccount">
    <constructor-arg index="0" value="20"/>
    <constructor-arg index="1" value="110-234-567890"/>
</bean>

<bean id="member" class="com.ohgiraffers.common.MemberDTO">
    <property name="sequence" value="1"/>
    <property name="name" value="홍길동"/>
    <property name="phone" value="010-1234-5678"/>
    <property name="email" value="hong123@gmail.com"/>
    <property name="personalAccount" ref="account"/>
</bean>
```

`<property>` 태그는 `setter` 메소드를 통해 빈 객체의 값을 초기화하는 설정이다.

- `name` : 필드명
- `value` : 필드에 담을 값
- `ref` : 참조할 빈의 id

`MemberDTO` 는 `Account` 타입을 의존하고 있기 때문에 `member` 빈 등록 시 `account` 빈을 참조하도록 `<property>` 태그의 `ref` 속성을 작성한다. Setter 메소드를 통해 의존성 객체를 전달하여 의존성을 주입하고 있으므로 이를 `세터 주입` 이라 한다.

▼ 실행 결과

```
PersonalAccount(bankCode=20, accNo=110-234-567890,
  balance=110-234-567890 계좌의 현재 잔액은 0원 입니다.)
110-234-567890 계좌의 현재 잔액은 0원 입니다.
10000원이 입금되었습니다.
110-234-567890 계좌의 현재 잔액은 10000원 입니다.
5000원이 출금되었습니다.
110-234-567890 계좌의 현재 잔액은 5000원 입니다.
```

빈 객체를 초기화 하는 방법이 생성자 또는 setter 메소드라는 차이는 있으나 테스트 코드의 결과는 동일하다.

2-2. Java Configuration

2-2-1. 생성자(Coustructor) 주입

```
@Bean
public Account accountGenerator() {

    return new PersonalAccount(20, "110-234-567890");
}

@Bean
public MemberDTO memberGenerator() {

    /* MemberDTO 생성자를 통해 Account를 생성하는 메소드를 호출한 리턴 값을 전달하여 bean을 조립할 수 있다. */
    return new MemberDTO(1, "홍길동", "010-1234-5678", "hong123@gmail.com", accountGenerator());
}
```

`MemberDTO` 는 `Account` 타입을 의존하고 있기 때문에 `memberGenerator` 빈 등록 시 `accountGenerator` 빈을 참조하도록 MemberDTO 생성자의 인자로 accountGenerator 메소드 호출의 결과(PersonalAccount bean 객체)를 전달한다. 생성자를 통해 의존성 객체를 전달하여 의존성을 주입하고 있으므로 이를 `생성자 주입` 이라 한다.

```
/* Java 설정 파일을 기반으로 ApplicationContext 객체 생성 */
ApplicationContext context
  = new AnnotationConfigApplicationContext(ContextConfiguration.class);

/* MemberDTO 타입의 빈 가져오기 */
MemberDTO member = context.getBean(MemberDTO.class);

/* MemberDTO의 PersonalAccount 객체 출력 */
System.out.println(member.getPersonalAccount());
/* 10000원 입금 */
System.out.println(member.getPersonalAccount().deposit(10000));
/* 잔액 출력 */
System.out.println(member.getPersonalAccount().getBalance());
/* 5000원 출금 */
System.out.println(member.getPersonalAccount().withDraw(5000));
/* 잔액 출력 */
System.out.println(member.getPersonalAccount().getBalance());
```

▼ 실행 결과

```
PersonalAccount(bankCode=20, accNo=110-234-567890,
  balance=110-234-567890 계좌의 현재 잔액은 0원 입니다.)
110-234-567890 계좌의 현재 잔액은 0원 입니다.
10000원이 입금되었습니다.
110-234-567890 계좌의 현재 잔액은 10000원 입니다.
5000원이 출금되었습니다.
110-234-567890 계좌의 현재 잔액은 5000원 입니다.
```

2-2-2. 세터(Setter) 주입

```
@Bean
public Account accountGenerator() {

    return new PersonalAccount(20, "110-234-567890");
}

@Bean
public MemberDTO memberGenerator() {

    MemberDTO member = new MemberDTO();
    member.setSequence(1);
    member.setName("홍길동");
    member.setPhone("010-1234-5678");
    member.setEmail("hong123@gmail.com");
    /* setter를 통해 Account를 생성하는 메소드를 호출한 리턴 값을 전달하여 bean을 조립할 수 있다. */
    member.setPersonalAccount(accountGenerator());

    return member;
}
```

`MemberDTO` 는 `Account` 타입을 의존하고 있기 때문에 `memberGenerator` 빈 등록 시 `accountGenerator` 빈을 참조하도록 setPersonalAccount 메소드의 인자로 accountGenerator 메소드 호출의 결과(PersonalAccount bean 객체)를 전달한다. setter를 통해 의존성 객체를 전달하여 의존성을 주입하고 있으므로 이를 `세터 주입` 이라 한다.

▼ 실행 결과

```
PersonalAccount(bankCode=20, accNo=110-234-567890,
  balance=110-234-567890 계좌의 현재 잔액은 0원 입니다.)
110-234-567890 계좌의 현재 잔액은 0원 입니다.
10000원이 입금되었습니다.
110-234-567890 계좌의 현재 잔액은 10000원 입니다.
5000원이 출금되었습니다.
110-234-567890 계좌의 현재 잔액은 5000원 입니다.
```

빈 객체를 초기화 하는 방법이 생성자 또는 setter 메소드라는 차이는 있으나 테스트 코드의 결과는 동일하다.

정리

DI는 객체 간의 `의존 관계`를 빈 설정 정보를 바탕으로 컨테이너가 자동적으로 연결해주는 것이다. 이를 통해 객체 간의 `결합도`를 낮출 수 있으며 이로 인해 `유지보수성`과 `유연성`이 증가한다.

- XML 빈 설정 시에는 `<constructor-args>` 또는 `<property>` 태그의 `ref` 속성에 의존성 주입할 bean의 이름을 설정한다.
- Java 빈 설정 시에는 `생성자`, `setter 메소드`의 인자 값으로 의존성 주입할 bean의 메소드 호출 반환 값을 전달한다.