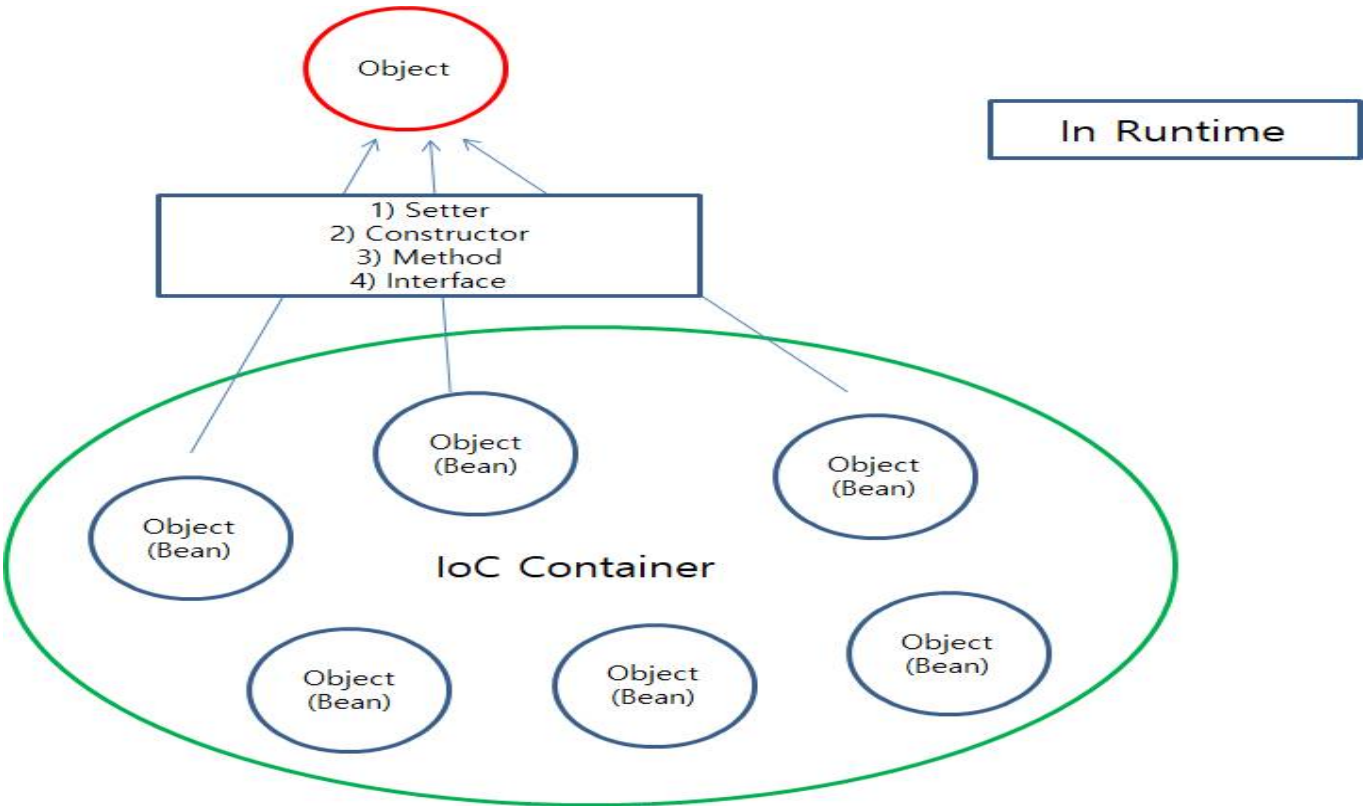
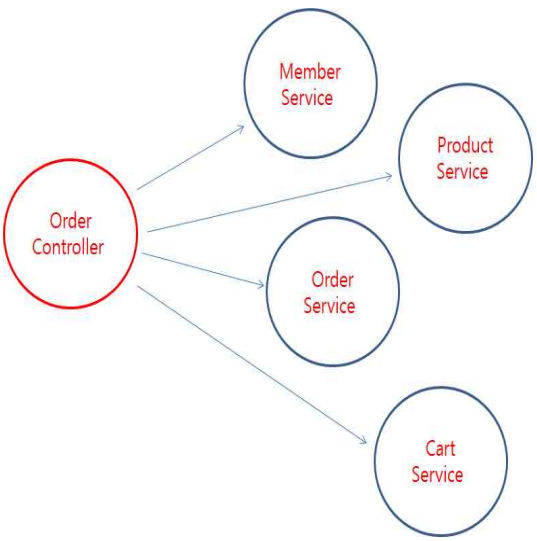
스프링 DI(Dependency Injection) & IoC(Inversion Of Control)

@) IoC(Inversion Of Control) Container

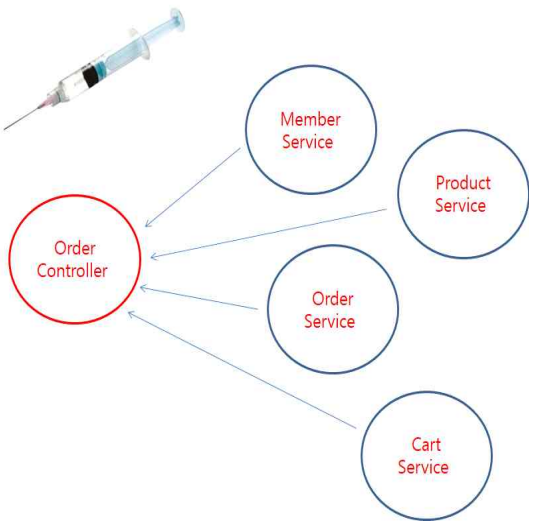


@DI(Dependency Injection)

AsIs Model



ToBe Model



@) IoC(Inversion of Control) , IoC Container

- 스프링 프레임워크는 객체에 대한 생성 및 생명주기를 관리할 수 있는 기능을 제공하고 있다.

- 제어권 역전(행)은 애플리케이션의 흐름을 사용자가 아닌 프레임워크가 관리하는 디자인 원칙이다.

- 전통적인 프로그래밍에서는 애플리케이션의 흐름을 사용자가 직접 관리하며 필요한 객체를 생성하여 의존성을 관리하였는데 IoC에서는 이러한 작업을 프레임워크가 대신 수행한다.

- 개발자는 비즈니스 로직에 더 집중할 수 있으며 코드는 더 깔끔하고 테스트 및 유지보수가 용이해진다.

@) 의존성주입(Dependency Injection)

- 의존성주입은 IoC의 한 형태로 객체가 필요로 하는 의존성을 외부에서 주입하는 기술이다.

- 개발자는 xml 파일이나 어노테이션을 통해 빈(Bean)과 의존성을 정의 할 수 있으며 IoC 컨테이너는 이 정보를 바탕으로 어플리케이션 실행시에 객체를 생성하고 의존성을 주입한다.

- IoC 컨테이너는 애플리케이션내의 객체들(빈)을 생성하고 이들사이의 의존성을 관리한다.

- 이를 통해 객체간의 결합도를 낮추고 유연성과 테스트 용이성을 향상시킨다.

- 스프링에서는주로 @Autowired 어노테이션을 사용하여 의존성을 자동으로 주입한다.

- 객체 생성과 객체 사용을 분리하여 이를 구현한다. 이는 SOLID의 의존관계 역행 및 단일 책임 원칙을 따르는 데 도움이 된다.

- 코드의 재사용성을 향상시킬수 있다.

- 클래스를 수정해야 하는 빈도를 줄이는 것을 목표로 한다.

- 의존성 주입은 객체 사용 생성을 분리하여 이러한 목표를 지원한다. 이를 통해 종속성을 사용하는 클래스를 변경하지 않고도 종속성을 바꿀 수 있다. 또한 의존성 중 하나가 변경되었기 때문에 클래스를 변경해야 하는 위험도 줄어든다.

- 의존성 주입 기술은 많은 최신 애플리케이션 프레임워크가 이를 구현한다. 이러한 프레임워크는 기술적 부분을 제공하므로 비즈니스 로직 구현에 집중할 수 있다.

@) IoC & DI 동작과정

[동작과정]

1) 구현클래스의 정보를 xml(설정파일)에 기술하거나 클래스에 어노테이션을 명시한다.
2) Framework(IoC Container)에서 설정파일을 읽거나 어노테이션을 읽어 들인다.
3) 구현객체를 생성하며 서로 의존관계가 있는지 확인하여 객체를 연결한다.

[주입방법]

1) 생성자(Constructor) 주입 : 필요한 의존성을 모두 포함하는 클래스의 생성자를 만들고 그 생성자를 통해 의존성을 주입한다.

2) 세터(Setter)를 통한 주입 : 의존성을 입력받는 세터(Setter) 메서드를 만들고 이를 통해 의존성을 주입한다.
3) 메서드(Method)를 통한 주입 : 의존성을 입력받는 메서드를 생성하여 의존성을 주입한다.
4) 인터페이스(Interface)를 통한 주입 : 의존성을 주입하는 함수를 포함한 인터페이스를 작성하고 이 인터페이스를 구현하도록 함으로써 실행시에 이를 통하여 의존성을 주입한다.

@) 인터페이스 도입

- 상위 클래스와 하위 클래스 간의 종속성을 끊기 위해 인터페이스를 도입할 수 있다. 그렇게 하면 두 클래스 모두 인터페이스에 의존하고 더 이상 서로 의존하지 않는다.

- 이는 상위 수준 클래스를 해당 종속성에서 분리하므로 이를 사용하는 코드를 변경하지 않고도 하위 수준 클래스의 코드를 변경할 수 있다. 종속성을 직접 사용하는 유일한 코드는 인터페이스를 구현하는 특정 클래스의 개체를 인스턴스화하는 코드이다.

- 인터페이스 역할을 건너뛰고 서비스 객체를 클라이언트에 직접 주입할 수 있지만 그렇게 하면 의존관계 역전 원칙이 깨지고 클라이언트는 서비스 클래스에 대한 명시적인 종속성을 갖게 된다. 어떤 상황에서는 이것이 괜찮을 수도 있지만 그러나 대부분의 경우 클라이언트와 서비스 구현 간의 의존성을 제거하기 위해 인터페이스를 도입하는 것이 더 좋다.