## **스프링 핵심 원리 - 기본편**

* 스프링부트란?
  + 스프링을 편하게 사용하도록 지원
  + 웹서버 내장
  + 손쉬운 빌드 구성을 위한 starter 종속성 제공
  + 스프링과 써드파티 라이브러리 자동 구성
  + 쉬운 설정
  + <https://start.spring.io/> 스프링부트 프로젝트 생성 사이트
* 스프링의 핵심
  + 자바 기반 프레임워크로 객체 지향(OOP)의 가장 강한 특징을 살림
  + 좋은 OOP를 개발하게 도와주는 프레임워크
* 객체 지향 특징
  + 추상화
  + 캡슐화
  + 다형성(핵심!)
    - client는 인터페이스만 알면됨
    - 구현대상의 내부 구조를 몰라도 됨
    - 내부구조의 변경에도 영향을 받지 않음
    - 구현 대상 자체를 변경해도 영향을 받지 않음
    - 객체 설계시 인터페이를 먼저 부여 후 구현 객체 생성
  + 상속
* 좋은 객체지향설계의 5가지 원칙(SOLID)
  + SRP : 단일책임의 원칙(한 클래스는 한 책임만)
  + OCP : 개방-폐쇄 원칙(확장O 변경X)
  + LSP : 리스코프 치환 원칙
    - 다형성에서 하위 클래스는 인터페이스 규약을 모두 지켜야함
  + ISP : 인터페이스분리원칙(범용하나보다 여러개의 인터페이스가 좋다)
  + DIP : 의존관계 역전 원칙(추상화에 의존, 구체화 의존X)
  + 다형성으로는 OCP,DIP를 지킬수 없어 스프링을 활용
* 스프링과 객체지향
  + IoC(Inversion of Control) : 제어의 역전
    - 프로그램의 제어 흐름을 외부에서 관리하는것
  + 프레임워크와 라이브러리
    - 프레임워크 : 내 코드를 제어하고 대신실행
    - 라이브러리 : 내 코드가 직접 제어하고 실행
  + DI(Dependencies injection) 의존관계 주입
    - 애플리케이션 실행 시점에 외부에서 실제 구현 객체를 생성하고 클라이언트에 전달, 클라이언트-서버 실제 의존관계가 연결
    - 동적인 클래스 의존관계 :
      * application실행시점에 생성된 객체 인스턴스의 참조가 연결된 의존관계
      * <->정석인 클래스 의존관계 : 클래스 내 코드로 보여지는 의존관계
  + IoC(DI)컨테이너
    - 객체의 생성과 관리하며 의존관계 연결
    - ApplicationContext를 Spring컨테이너라고도 함
    - 스프링 컨테이너는 @Configuration이 붙은 코드를 정보로 사용
      * @Bean이 적힌 메서드를 모두 호출하여 반환된 객체를 스프링컨테이너에 등록하고 이 객체를 스프링 빈이라고함
* 스프링 컨테이너와 스프링빈
  + 스프링 컨테이너 생성
    - ApplicationContext ac = new AnnotationConfigApplicationContext(파일명.class)
  + 스프링컨테이너 생성 및 빈 등록
    - Spring컨테이너 생성 시에는 파라미터로 구성정보를 받음
    - 파라미터로 넘어온 클래스 정보로 스프링빈을 등록
  + 설정정보를 참고하여 의존관계를 주입
  + 컨테이너에 등록된 빈조회(ApplicationContext)
    - getBeanDefinitionNames() : 스프링에 등록된 모든 빈 이름 조회
    - getBean() : 빈의 이름으로 빈객체를 조회
    - getRole() : 빈의 구분이 가능
      * ROLE\_APPLICATION : 사용자 정의한 빈
      * ROLE\_INFRASTRUCTURE: 스프링이 내부에서 사용하는 빈
* 싱글톤 컨테이너
  + 사용이유 : 여러요청이 동시에 들어오는데 요청마다 객체 생성시 낭비가 심함
  + 싱글톤 : 클래스의 인스턴스가 1개만 생성되도록 하는 디자인패턴
  + 문제점
    - 구현 코드가 많음
    - client가 클래스에 의존(DIP/OCP 위반)
    - 자식클래스 만들기 어려움
  + 싱글톤 컨테이너
    - 객체 인스턴스를 싱글톤으로 관리
    - 싱글톤 구현을 위한 코드가 줄어듬
  + 주의점 : 하나의 객체를 공유하기 떄문에 무상태로 설계해야함
    - 값 변경이 가능한 필드X
    - 가급적 읽기만
    - 필드대신 지역변수, 파라미터 사용
    - @configuration을 사용하고 @Bean을 사용해야 CGLIB기술을 통해 싱글톤 보장
    - @bean만 사용하면 빈으로 등록은 되지만 싱글톤 보장X
* 컴포넌트 스캔
  + 별도의 설정정보 없이 자동으로 스프링 빈을 등록하는 기능
  + 어노테이션
    - @Autowired: 빈의 의존관계를 자동주입하는 기능
    - @Component 어노테이션이 붙은 클래스르 스캔하여 스프링빈으로 등록
    - @Configuration은 @Component를 가지고있어 사용가능
    - @Component(‘지정할 이름”)을 통해 빈이름 직접 등록
  + 스캔대상
    - @Controller : 컨트롤러에서 사용
    - @Service : 비즈니스 로직에서 사용
    - @Repository : 데이터 접근계층에서 사용
    - @Configuration : 스프링설정정보에서 사용
  + 중복등록/충돌
    - 자동빈등록+자동 : 예외발생(ConflictingBeanDefinitionException)
    - 수동 + 자동 : 스프링에선 수동이 우선권//스플이부트에선 오류발생
* 의존관계 자동 주입
  + 생성자 주입 : 생성자 통해 의존관계 주입
    - 컴포넌트가 스프링빈에 등록할때 생성자 호출->컨테이너에서 의존관계 주입
    - 1번만 호출 보장(권장)
    - 생성자가 1개라면 @Autowired생략가능
  + 수정자 주입 : setter로 의존관계 주입
    - 선택, 변경 가능성 있는 의존 관계에 사용
  + 필드주입: 필드에 의존관계 주입
    - 외부에서 변경이 불가능해 테스트하기 힘듬
    - 테스트 코드X
  + 생성자 주입 선택 이유
    - 불변 : 대부분 application종료까지 의존관계 변경X
    - 누락 : 주입데이터 누락시 컴파일 오류
    - final : 필드에 final사용가능 값설정 없으면 컴파일 시점에 막아줌
  + 롬복 : 클래스에 @RequiredArgsConstructor를 붙여서 생성자 자동생성
  + 실무에서의 자동/수동등록
    - 자동기능을 권장
    - 수동등록 사용
      * DB연결이나 공통로그 처리처럼 업무로직 지원하는 기술
      * 비즈니스 로직에서 다형성을 적극 활용
* 빈 생명주기와 콜백
  + 스프링에서 객체의 초기화/종료 작업을 처리할때의 라이프사이클
  + 스프링컨테이너생성 -> 스프링빈생성 -> 의존관계 주입 -> 초기화 콜백

-> 사용 -> 소멸전 콜백 -> 스프링 종료

* + 데이터를 DI 끝난 다음에 사용가능하므로 초기화 작업은 DI 다음 호출
  + 초기화 콜백: 빈의 DI이 완료되고 호출
  + 소멸전 콜백: 소멸되기 직전 호출
  + 생성자 안에서 무거운 초기화 작업보다는 나누는것이 유지보수에 좋음
  + 인터페이스 방식 : 외부Library에 적용X 거의 사용안함
  + Bean설정 정보에 초기화/종료 매서드 지정
    - @Bean(initMethod=”매서드”, destroyMethod=”매서드”)
    - 스프링코드에 의존하지 않아 외부Library에도 적용가능
    - close, shutdown매서드를 자동으로 호출(배포주의)
  + 어노테이션 방식 : @PostConstruct / @PreDestroy
    - 가장 권장되고 편리, 스프링에 의존적이지 않은 자바표준
    - 해당 매서드에 직접 붙여줌
* 빈스코프
  + 빈이 존재 할수있는 범위
  + 기본적으로 싱글톤 스코프로 생성, 컨테이너의 시작과 끝까지 유지
  + 싱글톤스코프 : 기본스코프, 가장 넓은 범위의 스코프
  + 프로토타입 스코프: 컨테이너가 스코프의 생성과 DI까지만 관여하고 관리X
    - 매 사용마다 의존관계 주입이 완료된 새로운 객체가 필요하면 사용
    - 항상 새로운 인스턴스를 생성하여 반환
    - 관리 책임은 클라이언트가 가짐
    - PreDestroy같은 종료 매서드 호출X
    - 싱글톤스코프안에 프로토타입스코프가 사용될때 문제와 해결
      * 프로토타입빈 또한 싱글톤 스코프와 주입되서 끝나기 때문에 새로 생성되지 않고 싱글톤 스코프와 유지
      * 싱글톤 빈이 프로토타입 사용시 컨테이너에 새로 요청하게 변경
      * ObjectProvider : 빈을 컨테이너에서 대신 찾아주는 DL 서비스 제공
        + prototypeBeanProvider.getObject()를 통해 생성
        + 스프링에 의존
        + 권장
      * JSR-330 Provider
        + javax.inject.Provider라는 JSR-300 자바표준 사용
        + provider.get()을 통해 빈이 생성
        + 스프링에 의존X
  + 웹관련 스코프
    - request : 웹요청이 들어오고 나갈떄까지 유지, 요청마다 별도의 빈생성
      * 실제요청이 와야 생성되므로 지연처리가 필요
      * 해결1: provider을 통해 호출시점까지 지연처리
      * 해결2: proxyMode 통해 가짜 프록시 클래스를 만들고 요청시 위임
    - session : 웹세션이 생성되고 종료될떄까지 유지
    - application : 웹서블릿컨텍스와 같은범위로 유지
    - websocket : 웹소켓과 동일한 생명주기를 가짐
  + 클래스에 @Scope(“prototype/request”)등으로 등록