**IoC 1부**

1. 컨테이너 안에 들어있는 객체들을 빈(Bean) 이라고 부른다.
2. Ioc == 의존 관계 주입(DI), new를 통해 직접 생성하는 것이 아님.
3. @Service, @Autowired, @Repository 은 Bean에 등록되어 있는 객체

* @Repository라는 어노테이션을 붙이면 AutoScan으로 Bean 등록된다.

1. Spring IoC 최상위 객체는 Bean Factory

* 가장 핵심적인 클래스
* 가장 중요한 메섣 getBean()

1. Spring IoC 컨테이너

* Bean Factory
* 애플리케이션 컴포넌트의 중앙 저장소
* Bean설정 소스로부터 Bean 정의를 읽어들이고, Bean을 구성하고 제공.

1. Bean

* Spring IoC 컨테이너가 관리하는 객체
* 장점
  + 의존성 관리
  + 라이프사이클 인터페이스 지원
  + 스코프
* 싱글톤 : 하나만 만들어서 사용

Bean으로 등록할 때 아무런 어노테이션을 붙이지 않았다면 싱글톤으로 등록됨.

* 프로토타입 : 매번 다른 객체를 사용

1. Application Context

* Bean Factory에 비해 다양한 기능을 가지고 있는 것

e.g) 이벤트 발행, 리소스 로딩, 메시지 소스 처리

1. 의존성 주입을 받으려면 Bean으로 되어야 한다.
2. 싱글톤으로 만들어서 관리하고 싶은 객체는 IoC 컨테이너(Bean)으로 만들어 관리하면 편하다.
3. IoC에서 객체를 제공받는다면 해당 인스턴스는 항상 같은 객체.(메모리적을 효율적, Run time 시 성능 최적화, 비용적 유리(한 번만 만듦))
4. assertThat -> Unit Test에서 사용됨, 에러 검출

Type에 대한 안정성 얻을 수 있음.

1. Mock

* 실제 객체를 만들어 사용하기에 시간, 비용 등의 Cost가 높거나 혹은

객체 서로 간의 의존성이 강해 구현하기 힘들 경우 가짜 객체를 만들어 사용하는 방법.

IoC 2부

\*어노테이션(Annotation) : 다양한 목적이 있지만 메타 데이터의 비중이 가장 크다. (데이터에 대한 설명)

1. Spring boot로 프로젝트를 만들면 spring-boot-starter-web이 필요한 의존성을 기본 Setting 해준다.
2. Spring IoC 컨테이너는 Bean 설정파일이 있어야 한다.

* resources/application.xml (아주 고전적인 설정파일)

<bean …. <property … /> /> 이 방법은 매우 귀찮고 비효율적이기 때문에 <context:component-scan /> 을 이용하는 방법이 있음.

Java 파일로 Bean 등록을 할 때

*@ComponentScan*(basePackageClasses = DemoApplication.class)

를 사용하면 더욱 Type safe하고 편하게 등록 가능.

1. *@SpringBootApplication* 만 붙어 있다면 @ApplicationConfig 필요없다 (Spring Boot)

IoC 3부 - @Autowired

* Java의 Interface

: 추상메서드의 모음.

인터페이스를 만든다면 반드시 구현하는 클래스를 만들어야함.

인터페이스를 구현하기로 한 클래스는 반드시 인터페이스에 명시되어 있는 추상메서드들을 모두 구현해야 한다.

1. *@Primary*

* 중복 Bean 중 해당 어노테이션 Bean을 사용하겠다.
* 해당 방법을 추천.

1. Bean의 아이디는 첫 글자 SmallCse(소문자)의 Class name
2. BeanPostProcessor

* 빈의 인스턴스를 만든 다음 빈을 초기화 이전 혹은 이후에 부가적인 작업을 할 수 있는 또 다른 라이프사이클 콜백
* @PostConstruct의 어노테이션을 이용해 설정 가능

(빈이 만들어진 다음 해야 할 일)

빈이 이미 주입이 다된 상태

1. ApplicationContext -> 빈을 담고 있음.
2. 모든 빈은 등록되면 AutoWired로 꺼내서 쓸 수 있다.

Ioc 4부 - @Componet와 컴포넌트 스캔

1. @SpringBootApplication을 통해 Component Scan이 시작되고 다른 package까지는 Scan하지 못 한다.
2. 펑션을 사용한 빈 등록 (너무 불편하긴 함 > 일일이 등록)

* 리플렉션이나 프록시기반 cg라이브러리 기술을 사용하지 않음.
* 위 2가지는 성능에 영향을 주지만, 펑션을 사용하면 성능(애플리케이션 구동 타임)상에 이점은 있다.
* @Bean 으로 빈 등록을 펑션을 사용한 빈 등록으로 대체 가능.

1. BeanFactoryPostProcessor : 다른 모든 빈들을 만들기 이전에 적용.

* 다른 빈들(e.g) @Bean)을 만들기 이전에 컴포넌트 스캔을 해서 등록을 해 줌.

IoC 5부 – 빈의 스코프

1. 아무런 설정을 하지 않으면 기본 스코프는 싱글톤이다.

: 애플리케이션 전반에 걸쳐서 해당 빈의 인스턴스가 오직 한 개 뿐이다.

1. 거의 대부분의 경우에는 싱글톤 스코프만 씀.
2. *@Scope*("prototype")
3. ->빈을 받아올 때마다 새로운 인스턴스가 됨.
   1. proto
   2. com.example.demo.Proto@47dd778
   3. com.example.demo.Proto@15515c51
   4. com.example.demo.Proto@36a7abe1
   5. single
   6. com.example.demo.Single@64a896b0
   7. com.example.demo.Single@64a896b0
   8. com.example.demo.Single@64a896b0
4. *@Scope*(value = "prototype", proxyMode = *ScopedProxyMode*.***TARGET\_CLASS***)

* value = "prototype"
  + 싱글톤 사용하지 않고 빈을 받아올때마다 새로운 인스턴스 생성
* proxyMode = *ScopedProxyMode*.***TARGET\_CLASS***
  + 클래스 기반의 proxy로 감싸서 사용해라
* 프록시를 거쳐서 참조하도록 하라(프록시가 빈으로 등록됨)
* 직접 proto 를 쓰면 매번 바꿔줄 수 없으므로 매번 바꿔 주기 위해. (CG(Code Generator) 라이브러리 기반
* ObjectProvider<Proto> proto로 코드를 작성하면 같은 효과를 볼 수 있으나 코드에 직접적으로 Spring 코드가 들어가버림.

1. 싱글톤 객체 사용시 주의할 점
   1. 프로퍼티가 공유(int value = 0 과 같은 값이 멀티스레드에서 사용될 경우 서로 충돌할 수 있다. 값의 출력이 바뀔 수 있다, 스레드 세이프 코딩 필요)
   2. ApplicationContext 초기 구동 시 인스턴스가 생성된다.

(애플리케이션 구동 시 시간이 좀더 걸릴 수 있다.)

8부 – Environment 1부 프로파일

1. ApplicationContext 내 EnvironmentCapable
2. 프로파일 -> 빈들의 묶음(그룹).

테스트용, 실제 상용화 환경 등에 따른 빈들의 묶음.

1. 프로파일 표현식
   1. ! (not)
   2. & (and)
   3. | (or)

9부 – Environment 2부 프로퍼티

* 애플리케이션에 제공되는 key – value 쌍(OS 환경변수, 자바 애플리케이애플 실행할 때 넘겨주는 -d 옵션, 서블렛 Config)자바 의 프로퍼티에 접근할 수 있는 기능.
* 계층형으로 접근

11. ApplicationEventPublisher

-