Instantiation Beans

Bean Definition은 본질적으로 하나 이상의 객체를 생성하기 위한 레시피입니다.

컨테이너는 빈 객체 생성 요청 시에 명명된 빈의 레시피를[^1] 확인하고, 해당 빈 정의에 캡슐화된 구성 메타데이터(BeanDefinition의 속성에 저장됨)를 사용하여 실제 객체를 생성(또는 획득)합니다.

> 해당 빈 정의에 캡슐화된 구성 메타데이터란?
>
> "해당 빈 정의에 캡슐화된 구성 메타데이터"는 빈을 생성하는 데 필요한 모든 정보를 담고 있으며, 이 정보는 스프링에서 BeanDefinition 객체의 속성들에 저장됩니다.
>
> **BeanDefinition 객체의 주요 속성들**
> BeanDefinition 객체는 스프링 컨테이너가 빈을 생성하고 관리하기 위해 사용하는 메타데이터를 포함합니다. 여기에는 다음과 같은 주요 속성들이 포함됩니다:
> 1. 빈 클래스 이름 (Bean Class Name):
> - 빈이 어떤 클래스의 인스턴스인지를 정의합니다. 스프링 컨테이너는 이 클래스 정보를 사용하여 빈의
> 인스턴스를 생성합니다.
>
> 2. 스코프 (Scope):
> - 빈의 범위를 정의합니다. 예를 들어, **singleton**, **prototype**, request, session 등의 스코프가 있습니다.
> singleton 스코프는 빈의 단일 인스턴스가 애플리케이션 컨텍스트에 존재하도록 하고,
> prototype 스코프는 요청할 때마다 새로운 인스턴스를 생성합니다.
>
> 3. 생성자 인자 값 (Constructor Arguments):
> - 빈이 생성될 때 생성자에 전달되는 인자들을 정의합니다. 이를 통해 스프링은 올바른 생성자를 호출하여
> 객체를 생성할 수 있습니다.
>
> 4. 프로퍼티 값 (Property Values):
> - 생성된 빈의 속성(property) 값을 설정하는데 사용됩니다.
> 이는 setter 메서드나 필드에 값을 주입하는 데 사용됩니다.
>
> 5. 의존성 (Dependencies):
> - 빈이 생성될 때 필요한 다른 빈에 대한 의존성을 정의합니다. 이를 통해 스프링은 필요한 다른 빈들을
> 먼저 생성하고, 이 빈에 주입할 수 있습니다.
>
> 6. 초기화 메서드 (Init Method):
> - 빈이 생성된 후 호출될 초기화 메서드를 정의합니다.
>
> 7. 소멸 메서드 (Destroy Method):
> - 빈이 컨테이너에서 제거될 때 호출될 메서드를 정의합니다.
>
> **예시: BeanDefinition 메타데이터 예시**

```
@Configuration
public class AppConfig {

    @Bean(initMethod = "init", destroyMethod = "destroy")
    public MyService myService() {
        MyService myService = new MyService("Hello, Spring!");
        myService.setDependency(dependencyBean());
        return myService;
    }

    @Bean
    public DependencyBean dependencyBean() {
        return new DependencyBean();
    }
}
```

위의 예시에서, **myService** 빈은 다음과 같은 메타데이터를 가지게 됩니다:

- 빈 클래스 이름: MyService
- 스코프: singleton (기본값)
- 생성자 인자 값: 없음 (기본 생성자 사용)
- 프로퍼티 값: setDependency 메서드를 통해 주입된 DependencyBean
- 초기화 메서드: init
- 소멸 메서드: destroy
- 의존성: dependencyBean

이 모든 정보는 **BeanDefinition** 객체의 속성으로 캡슐화되며, 스프링 컨테이너는 이 정보를 사용하여 실제 빈 객체를 생성하고 관리합니다.

따라서, "해당 빈 정의에 캡슐화된 구성 메타데이터"는 바로 이 **BeanDefinition** 객체의 속성들에 저장된 정보들을 의미합니다. 이 정보들은 빈을 생성, 초기화, 주입, 관리하는 데 필요한 모든 것을 포함하고 있습니다.

XML 기반 구성 메타데이터를 사용하는 경우, 인스턴스화할 객체의 타입(또는 클래스)을 엘리먼트의 **class** 속성에 지정합니다. 이 **class** 속성(내부적으로는 **BeanDefinition** 인스턴스의 **Class** 속성)은 일반적으로 필수입니다. (예외 사항에 대해서는 Instantiation by Using an Instance Factory Method 및 Bean Definition Inheritance 을 참조하십시오.) **Class** 속성은 다음 두 가지 방법 중 하나로 사용할 수 있습니다:

- 일반적으로 컨테이너 자체가 생성자를 **반사적으로 호출**하여 Bean을 직접 생성하는 경우 생성될 Bean 클래스를 지정하는 것은 new 연산자를 사용하는 Java 코드와 다소 동일합니다.("**반사적으로 호출**한다"는 의미는 자바 리플렉션(Reflection) API를 사용하여 런타임에 클래스의 생성자를 호출하는 것을 말합니다. 즉, 컴파일 시점이 아닌 런타임 시점에 클래스의 메타데이터(예: 클래스, 생성자, 메서드, 필드 등)를 조사하고, 이를 통해 객체를 생성하거나 메서드를 호출하는 것을 의미합니다.)
- 컨테이너가 빈을 생성하기 위해 특정 클래스의 static 팩토리 메서드를 호출하는 조금은 일반적이지 않는 경우, 객체를 생성하기 위해 호출되는 static 팩토리 메서드가 포함된 실제 클래스를 지정합니다. static 팩토리 메서드 호출에서 리턴된 객체 타입은 동일한 클래스이거나 완전히 다른 클래스일 수 있습니다.

```
// MyService.java
public class MyService {
```

```
    private String message;

    // private 생성자
    private MyService(String message) {
        this.message = message;
    }

    // static 팩토리 메서드
    public static MyService createInstance() {
        return new MyService("Hello from MyService!");
    }

    public String getMessage() {
        return message;
    }
}

// AppConfig.java (스프링 설정 클래스)
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class AppConfig {

    @Bean
    public MyService myService() {
        return MyService.createInstance();
    }
}
```

> Nested class names
> 중첩 클래스에 대한 빈 정의를 구성하려면, 중첩 클래스의 바이너리 이름이나 소스 이름을 사용할 수 있습니다.
>
> 예를 들어, com.example 패키지에 SomeThing이라는 클래스가 있고, 이 SomeThing 클래스에 OtherThing이라는 정적 중첩 클래스가 있는 경우, 이들은 달러 기호($) 또는 점(.)으로 구분할 수 있습니다. 따라서 빈 정의의 class 속성 값은 com.example.SomeThing$OtherThing 또는 com.example.SomeThing.OtherThing이 됩니다.

[^1]: 레시피란 Bean Definition이 객체를 어떻게 생성할지, 어떤 속성을 설정할지, 어떤 의존성을 주입할지를 명시한 설명서 또는 설계도와 같은 역할을 한다는 것을 비유적으로 표현한 것입니다.