Video 05 Spring Boot: Bean and its Lifecycle | Inversion of Control (IOC).

What is Bean ?

Bean is a Java Object, which managed by Spring Container (also known as IOC container)

IOC → contains all beans which get created and also manages them.

(@Component Annotation , @Bean annotation) → Ways in which Bean can be created.

1. @Component.

```
@Component
public class User {

    String username;
    String email;

    getter() & setter()
}
```

@Component → It tells Spring Boot you have to create its Object and you have to manage its lifecycle.

→ Spring Boot will internally call new User()

[@Controller, @Service] → All internally use @Component

↓ Creates ~~the~~ Bean and Manages it.

Checkout this example

```
@Component
public class User {

    String username;
    String email;

    public User(String username, String email){
        this.username = username;
        this.email = email;
    }
```

[Here we have manually given the constructor. So Java will not give its default constructor.]

So now @Component will fail and it will not be able to create the object of User.

```
}
```

~~xxx~~ (Imp)

This where @Bean can help

with using @Bean we can provide configurations and give defaults what SpringBoot needs to use when creating a Bean.

User class ⇒ Same as mentioned above,

{If using @Bean annotation then @Component above the class is not required}

```
@Configuration.
public class AppConfig{

    @Bean
    public User createUserBean(){
        return new User("abc","xxx")
    }
}
```

Important Case w.r.t @Bean & @Component

Lets say we used both @Component & @Bean.
Both @Component & @Bean both are present

```
@Component
public class User{

    String username;
    String email;

    public User(){
                    -> default constructor.
    }

    public User(String username, String email){

        this.username = username;      ↓
        this.email = email;   argument
                              constructor.
    }

}
```

```
@Configuration
public class AppConfig{

    @Bean
    public User createUserBean(){
        return new
            User(" ", " ");
    }.
```

In the scenario where both are present @Bean, @Component.

What is to be done?

@Bean annotation gains preference. Even though @Component and default constructor is there still @Bean and paramaterised constructor will be called.
-> More Preference

Another example

```
@Configuration
public class AppConfig {

    @Bean
    public User createUserBean () {
        return new User ("abc", "abc.gmail");
    }

    @Bean
    public User createUserBean2() {
        return new User ("efg", "efg.gmail");
    }
}.
```

In this case two objects will be created.

Spring will Create 2 beans of User.

How Spring Boot find these Beans?

1- Method 1 is by using @ComponentScan, it will scan specified packages and sub-packages.

It looks for all the classes with @Component, @Service, @Repository

```
@SpringBootApplication.
@ComponentScan (basePackages = "com.conceptandcoding.learningSpringBoot")
public class SpringBoot Application {

    psvm(){
    }
}.
```

2. Method 2 is by looking at any @Bean annotation present in @Configuration class.

```
@Configuration
  public class AppConfig {

      @Bean
      public User createUserBean(){

          return new User("==", "._");

      }
  }
```

Also @SpringBootApplication → @Configuration auto Configuration mentioned in it

It does component scan on all the files where the ~~main~~ package is for main method.

At what times, these beans get Created.

Eagerness

→ Some Beans get created, when we start our application.

→ Beans with Singleton Scope are eagerly initialised.

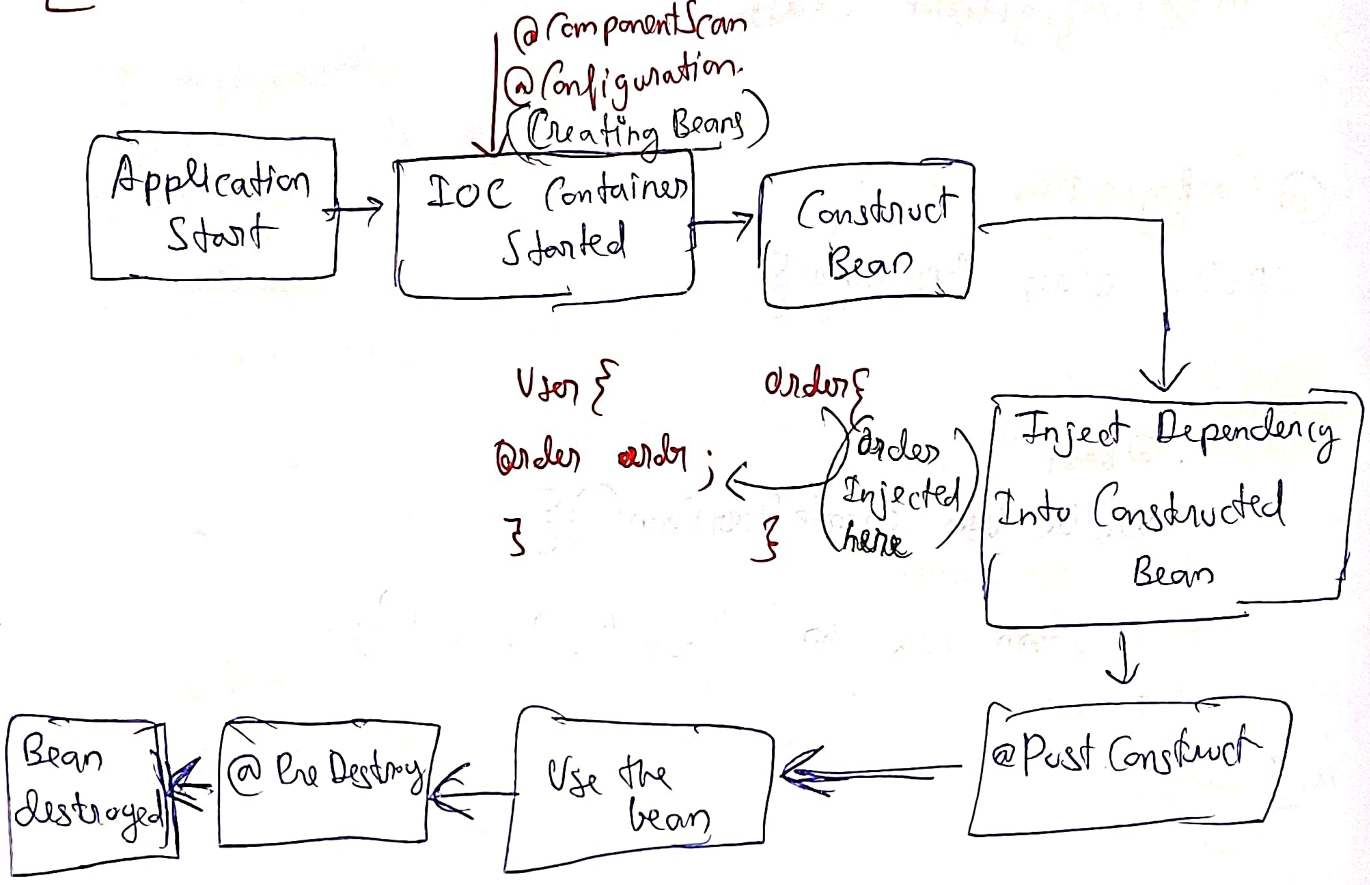
Lazy

→ Some Beans are created Lazily, means when they are actually needed

(Scope is Prototype)

→ ~~@Prototype~~ /@Lazy are Lazily created.

Life cycle of a Bean



During Application Startup, Spring Boot invokes IOC

IOC container, makes use of Configuration and @ComponentScan to lookout for the classes

In logs you will see — Invoking IOC container.

Initializing Spring embedded WebApplicationContext

Root WebApplicationContext: initialization completed in 419 ms.

Implementation of IOC logic.

(Inject Dependency Step)

```
@Component
public class User{

  @Autowired
  Order order;

  public User(){
     (Initializing user);
  }

}
```

Lazily

invoked into User class.

```
@Lazy
@Component
public class Order{

  public Order(){
     (initializing order);
  }
}
```

[illegible] *** Clearly note the Life cycle of a Bean diagram. Watch how it constructs the bean first and then dependency is injected.

So. First

1. initializing user.
2. initializing order

→ In this manner beans will be created