6. Spring boot: Bean and its Lifecycle | Inversion of Control (IOC)

1. **What is bean?**

In a Simple term, Bean is java Object, which is managed by Spring container [also known as IOC Container]

IOC container Contains all the beans which get created and also managed them.

2. **How to create Bean ?**

- @Component Annotation
- @Bean Annotation

Using @Component Annotations.

- it's follows **convention over configuration approach.**
- Means Spring boot will try to auto configure based on conventions reducing the need for configuration.
- @Controller, @Service etc. all are internally tells Spring to create bean and manage it.
- it uses default constructor for creating internal beans.

So here, Spring boot will internally call new User() to create an instance of this class.

```
@Component
public class User {
    String username;
    String email;

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getEmail() {
```

```
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
}
```

```
package org.springframework.stereotype;

import java.lang.annotation.Documented;
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Indexed
public @interface Component {
    String value() default "";
}
```

```
@Component
public class User {
    String username;
    String email;

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

    public void setUsername(String username) {
        this.username = username;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.e
```

****

APPLICATION FAILED TO START

****

- Here what happened @Component eager to default constructor for creating internal bean but it's failed because we already provided a main parametrized constructor.
- Spring boot does not know to pass in constructor parameters .

So what to do ?

@Bean comes into the picture

- Where we provide the configuration details and tells Spring boot to use it while creating bean.

Remove @Component Annotations.

```
public class User {
    String username;
    String email;

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

    public void setUsername(String username) {
        this.username = username;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
}
```

```
package codesake.in.securecapita.config;

import codesake.in.securecapita.dto.User;
import org.springframework.context.annotation.Bean;
```

```
import org.springframework.context.annotation.Configuration;

//learning purpose only
@Configuration
public class UserConfig {
    @Bean
    public User createUserBean(){
        return new User("saurav saxena","sauravsrivastava121@gmail.com");
    }
}
```

 If we are providing @Component notation On User class and providing default as well parameter constructor and configure it in **UserConfig Spring boot always gives priority to Configuration.**

In this below condition spring boot create two bean, Spring boot created n numbers of Bean depends up on configuration.

Managed by **IOC**

```
@Configuration
public class UserConfig {
    @Bean
    public User createUserBean(){
        return new User("saurav saxena","sauravsrivastava121@gmail.com");
    }

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

Now, we know what is bean and how it's getting Created. But few Questions comes to our minds:

How Spring Boot find these Beans?

At what time, these beans get created.

How Spring boot find these Beans?

- Using @ComponentScan annotation. it will scan the specified package and sub-package for classes annotated with @Component, @Services etc.

```
@SpringBootApplication(exclude = {SecurityAutoConfiguration.class})
@ComponentScan(basePackages = "codesake.in.securecapita")
@EnableAsync
```

```
public class SecurecapitaApplication {

    private static final int STRENGTH = 12;
    public static void main(String[] args) {
       SpringApplication.run(SecurecapitaApplication.class, args);
    }

    @Bean
    public BCryptPasswordEncoder passwordEncoder(){
       return new BCryptPasswordEncoder(STRENGTH);
    }

}
```

- @Configuration is nothing but @Component itself.
- Through Explicit defining of bean via @Bean annotations in @Cofiguration class.

```
//
// Source code recreated from a .class file by IntelliJ IDEA
// (powered by FernFlower decompiler)
//

package org.springframework.context.annotation;

import java.lang.annotation.Documented;
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
import org.springframework.core.annotation.AliasFor;
import org.springframework.stereotype.Component;

@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Component
public @interface Configuration {
    @AliasFor(
        annotation = Component.class
    )
    String value() default "";

    boolean proxyBeanMethods() default true;

    boolean enforceUniqueMethods() default true;
}
```

At what time beans get created.

Eagerness

- Some beans get created, when we start the application.
- ex:- Beans are Singleton Scope Eagerly initialized.

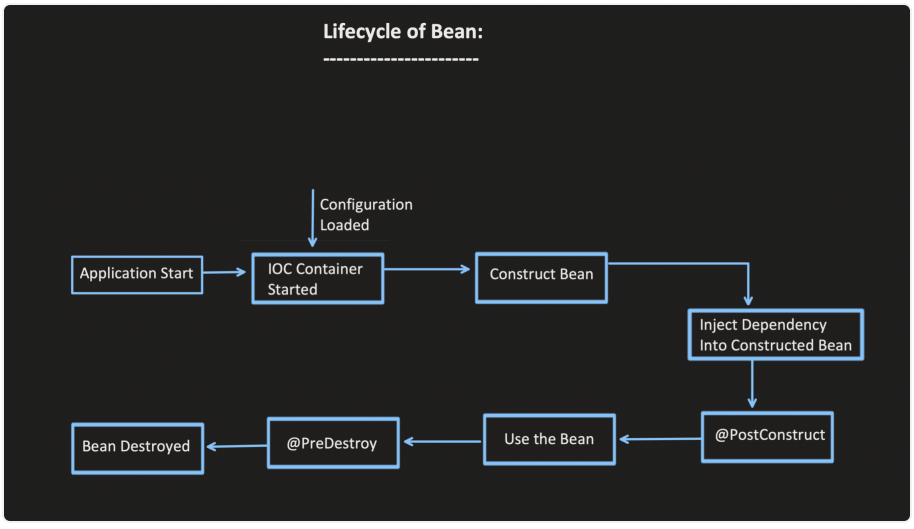
Lazy

- Some Beans get created Lazily means when they actually needed.
- ex:-Beans with like Prototype etc. are Lazily initialized. or Beans with @Lazy annotations.
- Even below is example of Singleton Scope but are annotate with @Lazy.
- If we are not providing @Scope and @Lazy then it's default singleton

```
@Lazy
@Component
public class Order{
    public Order(){
        System.out.println("Lazy Initialization");
    }
}
```

Life Cycle Of Bean.

Application Start IOC Container Started [Configuration Loaded] Construct Bean Inject Dependency Into Constructed Bean @PostConstruct Use the Bean @PreDestroy Bean Destroyed.



Step1:

- During Application Start- Up, Spring Boot invokes IOC Container.
- ApplicationContext provides the implementation of IOC Container.
- IOC Container, make use of Configuration and @ComponentScan to look out for the classes for which bean needed to be created.

```
Tomcat initialized with port 9090 (http)
Starting service [Tomcat]
Starting Servlet engine: [Apache Tomcat/10.1.20]
Initializing Spring embedded WebApplicationContext
Root WebApplicationContext: initialization completed in 5243 ms
HHH000204: Processing PersistenceUnitInfo [name: default]
```

Step 2:

Construct the Beans

Singleton Object will created at this point of time.

```
@Component
public class User{
    public Order(){
        System.out.println("Initialization User");
    }
}
//Initialization User
```

Step 3:

- Inject the Dependency info the constructed Bean.
- @Autowired, first look for bean of the required type

 If bean found, Spring will inject it. Different ways of injection:

- Constructor Injection.
- Setter Injection
- Field Injection

We will see each injection and which one to use in next part **********

 If bean is not found, Spring will create onw and the inject it.

```
@Component
public class User{
    @Autowired
    Order order;
    public Order(){
        System.out.println("Initialization User");
    }
}
```

```
@Lazy
@Component
public class Order{
    public Order(){
        System.out.println("Lazy Initialization order");
    }
}
//Initialization User
//Lazy Initialization order
```

Step 4: Perform any task before Bean to be used in application.

```
@Component
public class User{
    @Autowired
    Order order;

    @PostConstruct
    public void initialize(){
      System.out.println("Bean has been constructed and dependencies have been
injected");
    }

    public User(){
        System.out.println("Initialization User");
    }
}
//Initialization User
//Lazy Initialization order
//Bean has been constructed and dependencies have been injected
```

Step 5. Use the Bean in Your application.

Means, we are using the bean [or Object] to invoke some methods to perform some business logic.

Step 6:

Perform any task before Bean is getting destroyed.

- Here context.close() means we are forcefully close the IOC Container
- So, All beans will be destroyed.
- @preDesrtoy method get's invoked.
- ultimately IOC get's closed.

```
@SpringBootApplication
@EnableAsync
public class SecurecapitaApplication {
    public static void main(String[] args) {

        ConfigurableApplicationContext context =
SpringApplication.run(SecurecapitaApplication.class,args);
        context.close();
    }
}
```

```
@Component
public class User{
    @Autowired
    Order order;

    @PostConstruct
    public void initialize(){
      System.out.println("Post Construct Initiated");
    }

    @preDestroy
    public void preDestroy(){
      System.out.println("Bean is about to destroy, in Predestroyed Method");
    }

    public User(){
        System.out.println("Initialization User");
    }
}
//Initialization User
//Post Construct Initiated
//Bean is about to destroy, in Predestroyed Method
```