🍃

Section 4: What are Beans and How are they managed in Spring boot?

🍃In spring, bean is a simple object that is managed by the spring container. Instead of manually creating the object by using the "new" key word Spring automatically creates and manages them for us.

🍃 Now what does it actually mean to manage the bean?

💠Spring will create the object for you.

💠Spring will also keep the track of the object life cycle.

💠Spring will inject dependency if needed.

💠Spring also ensures that only one instance (singleton by default) is used wherever required.

💡 Think like beans to be the workers at the factory where they are assigned specific task. Workers are sent where ever the is work that needs to be done.

🍃How to define beans?

✅ There are two main ways of defining the beans in spring boot framework:

2️⃣Using beans inside the configuration class:

We have already seen what is the configuration class in previous section. To summarize it here it is just the manual configuration that we choose for our application instead of going with the spring default auto configuration. And as we need to let the spring boot framework know that "hey this is something i want to give it by myself and do not want to go default so please use this configuration class annotated by @Configuration.

So this configuration class can be used to define beans:

```
@Configuration
public class AppConfig{

    @Bean
    public MyService myservice(){
        return new MyService();   //spring will manage this object

        }
 }
```

What will happen here is as follows:

✅Spring will know that this is the configuration class and this will be scanned as during application run time as @SpringBootApplication annotation will run.

✅@Bean will tell hey manage this below object for me.

✅Spring will make sure to automatically provide the object of MyService where ever it is needed.

Instead of defining beans manually we can mark classes with special annotations:

@Service → For writing the business logic of the application.

@Repository → To write the query on the data in the data base.

@Controller → To manage the incoming HTTP requests.

There can be a scenario where we need some beans or objects or their references in other classes. Then how can we use them in other classes and inject them wherever necessary?

🍃The answer to this is by using the @Autowired annotation.

```
@Component
public class MyService {
    public void serve() {
        System.out.println("Service is running...");
    }
}

@Component
public class MyController {

    @Autowired  // Injecting MyService bean automatically
    private MyService myService;

    public void handleRequest() {
        myService.serve();
    }
    }
```

1️⃣Here Spring is detecting Myservice and MyController class as components and they are getting scanned by @ComponentScan **(which is included inside @SpringBootApplication annotation)** during the boot strapping. This is happening because we have marked them as @Component.

2️⃣Now lets say that we need the method of Myservice class in MyController class. So what will we do is we will declare the reference of the MyService class in MyComponent class as “myService” (m is small in the reference).

3️⃣Then we will annotate it with the @Autowired annotation and hence spring will do its job and let the Component class know that this is the reference of MyService class and now we can use that reference to access the methods and member functions of that class.

> Now we know that how the autowired annotation works.