1.3 Beans

What Are Beans in Java Spring Boot?

- In Spring Boot, a *Bean* is simply an *object* that is *instanced*, *assembled* and *managed* by the Spring IoC (Inversion of Control) container of Spring automatically, so we don't have to handle their lifecycle manually.
- Beans are the backbone of a Spring application and are the *core building blocks* that are *wired together* to create the application.

Spring Annotations:

In *traditional Spring Framework* applications, *beans* were typically defined and configured using XML configuration files. Developers had to explicitly declare each bean and its dependencies within these XML files.

However, with the introduction of Spring Boot, this process has been significantly simplified. Spring Boot promotes a *Java-based configuration approach* allowing to manage *bean component* programmatically. That's why *spring annotation* are introduced.

Ways to Define Beans:

1. **Using stereotype annotation:** Spring provides several annotations that automatically register a class as a bean within the Spring Application Context.

   Annotate the class with one of the stereotype annotations. This annotation informs spring that the class should be managed as a bean.

Annotation	Purpose
@Component	Generic stereotype for any component
@Service	Marks a class as a service layer bean
@Repository	Indicates a DAO (data access) component
@Controller	Marks a class as a web controller
@RestController	*@Controller* + *@ResponseBody* (REST APIs)

Main method (starting of SpringBoot application)

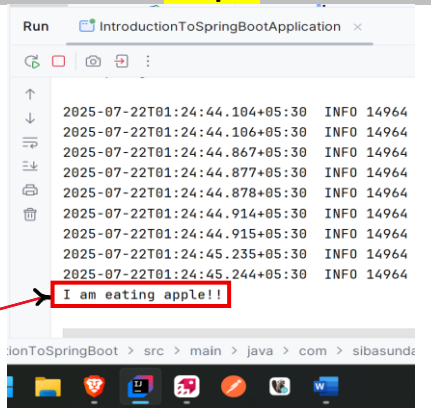
```
package com.sibasundar.week1Introduction.introductionToSpringBoot;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class IntroductionToSpringBootApplication implements CommandLineRunner {
    @Autowired
    private Apple apple;

    public static void main(String[] args) {
        SpringApplication.run(IntroductionToSpringBootApplication.class, args);
    }
    @Override
    public void run(String[] args) throws Exception {
        apple.eat();
    }
}
```

Output



Apple

```
package com.sibasundar.week1Introduction.introductionToSpringBoot;

import org.springframework.stereotype.Component;

@Component
public class Apple {
    public void eat() {
        System.out.println("I am eating apple!!");
    }
}
```

As we can see here *Apple* object is not instantiated yet but **eat()** method runs fine because of *Spring bean*. The *@Component* annotation tells Spring that the class is a Spring-managed bean while the *@Autowired* is used to automatically inject a Spring-managed bean into another bean.

2. **Explicit Bean Declaration in the Configuration class:** In Spring Boot, you can explicitly define beans using the `@Bean` annotation inside a class annotated with `@Configuration`.

   Create a *configuration class* and *annotate* it with *@Configuration*. This class will contain **methods** to define and configure beans.

Main method (starting of SpringBoot application)

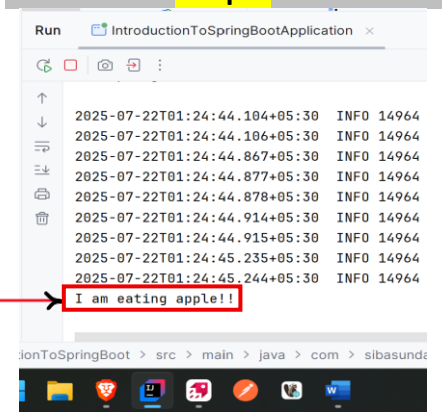
```
package com.sibasundar.week1Introduction.introductionToSpringBoot;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class IntroductionToSpringBootApplication implements CommandLineRunner {
    @Autowired
    private Apple apple;

    public static void main(String[] args) {
        SpringApplication.run(IntroductionToSpringBootApplication.class, args);
    }
    @Override
    public void run(String[] args) throws Exception {
        apple.eat();
    }
}
```

Output



Apple

```
package com.sibasundar.week1Introduction.introductionToSpringBoot;

import org.springframework.stereotype.Component;

public class Apple {
    public void eat() {
        System.out.println("I am eating apple!!");
    }
}
```

Configuration Class

```
package com.sibasundar.week1Introduction.introductionToSpringBoot;

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

@Configuration
public class AppConfig {

    @Bean
    Apple getApple() {      // it's like a factory method
        return new Apple();
    }
}
```

When we declare a bean using the ***@Bean*** annotation in a ***@Configuration*** class, we have the **full control** of how the **object** (bean) is created. That means we can **pass parameters**, **set properties**, or even **inject other beans** manually before returning the object.

Use which & when:

Stereotype annotation	Explicit Bean declaration
• Most application services • Repositories • Controllers • Clean architecture with auto-wiring	• Complex objects • Beans requiring runtime configuration • External classes (e.g., from libraries) • Testing-specific beans

Use *@Component* when it's your own class and needs **no special setup**.
Use *@Bean* when you **need customization**, or you **can't annotate** the class.

Bean Lifecycle:

Bean Created	The **bean instance** is created by invoking a **static factory method** or **an instance factory method** (for annotation-based configuration).
Dependency Injected	Spring sets the **bean's properties** and **dependencies**, either through ***setter injection***, ***constructor injection***, or ***field injection***.
Bean Initialized	If the bean implements the ***InitializingBean interface*** or defines a ***custom initialization*** method annotated with ***@PostConstruct***, Spring invokes the initialization method after the bean has been configured.
Bean is Used	The bean is now ***fully initialized*** and ready to be used by the application.
Bean Destroyed	Spring invokes the ***destruction method*** when the bean is no longer needed or when the ***application context*** is being ***shut down***.

Bean Lifecycle Hooks:

- The *@PostConstruct* annotation is used to **mark a method** that should be invoked **immediately after** a **bean** has been ***constructed*** and all of its ***dependencies*** have been ***injected***.
- The *@PreDestroy* annotation is used to **mark a method** that should be invoked **just before** a **bean** is ***destroyed*** by the **container**. This method can perform any necessary **cleanup** or **resource releasing** tasks.

Scope of Beans:

Scope	Description
singleton	(Default) Scopes a **single bean** definition to a **single object instance** for each **Spring IoC container**.
prototype	Scopes a **single bean** definition to any number of **object instances**.
request	Scopes a **single bean definition** to the **lifecycle** of a **single HTTP request**. That is, each **HTTP request** has its **own instance** of a bean created off the back of a single bean definition. Only valid in the context of a **web-aware Spring ApplicationContext**.
websocket	Scopes a **single bean definition** to the **lifecycle** of a **WebSocket**. Only valid in the context of a web-aware Spring ApplicationContext.