Destroy

In Spring, when you define beans using Java configuration (`@Configuration` annotated classes), Spring automatically manages the lifecycle of these beans. This includes creating the beans when they are needed and also destroying them when they are no longer needed, typically when the Spring application context is being shut down.

When Spring initializes a bean, it checks if the bean class has a public method named `close()` or `shutdown()`. If such a method exists, Spring assumes that it's a shutdown method for the bean and registers it as a destruction callback. This means that when the Spring application context is shutting down, Spring will automatically call this method on each bean that has it defined, allowing the bean to perform any necessary cleanup or shutdown tasks.

For example, consider the following bean definition:

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

@Configuration
public class MyConfiguration {

    @Bean
    public MyBean myBean() {
        return new MyBean();
    }
}
```

If the `MyBean` class has a method named `close()` or `shutdown()`, Spring will automatically register it as a destruction callback. When the Spring application context is being shut down, Spring will call this method on the `MyBean` instance.

However, there might be cases where you don't want Spring to automatically call the `close()` or `shutdown()` method on your beans during shutdown. In such cases, you can use the `destroyMethod` attribute of the `@Bean` annotation to specify an alternative method name or an empty string (`""`) to disable the default behavior.

For example:

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

```
@Configuration
public class MyConfiguration {

    @Bean(destroyMethod = "")
    public MyBean myBean() {
        return new MyBean();
    }
}
```

In this case, even if `MyBean` has a `close()` or `shutdown()` method, Spring will not call it when shutting down the application context because the `destroyMethod` attribute is set to an empty string (`""`). This gives you more control over the lifecycle management of your beans in Spring.

The public method (e.g., `close()` or `shutdown()`) appears in your bean class, typically as part of the bean's implementation. When defining a bean in Spring, if Spring detects that the bean class has a public method with a specific name (such as `close()` or `shutdown()`), it will automatically register that method as a destruction callback.

Here's an example:

Suppose you have a bean class named `MyBean`:

```java
public class MyBean {

    // Public method for cleanup or shutdown
    public void close() {
        // Perform cleanup tasks here
        System.out.println("Closing MyBean...");
    }
}
```

When you define a bean of type `MyBean` in your Spring configuration:

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

@Configuration
public class MyConfiguration {
```

```
    @Bean
    public MyBean myBean() {
        return new MyBean();
    }
}
```

Spring will automatically detect the `close()` method in the `MyBean` class and register it as a destruction callback. This means that when the Spring application context is being shut down, Spring will call the `close()` method on the `MyBean` instance, allowing it to perform any necessary cleanup or shutdown tasks.

In summary, the public method appears in your bean class, and Spring automatically detects and registers it as a destruction callback when defining beans in your Spring configuration.