🚀 `@PostConstruct` & `@PreDestroy` – Spring Bean Lifecycle Callbacks

Spring provides **lifecycle callbacks** to execute methods **before** and **after** a bean is used. The two key annotations for this are:

🔷 `@PostConstruct` (Runs After Bean Creation)

- Executes **just after** the bean is created and dependencies are injected.
- Useful for **initializing resources** or **preparing data**.

🔷 `@PreDestroy` (Runs Before Bean is Destroyed)

- Executes **just before** the bean is removed from the container.
- Useful for **closing resources** (e.g., database connections, file handles).

1️⃣ Student.java (Uses Address Bean & Lifecycle Callbacks)

```
java
CopyEdit
package in.sp.beans;

import jakarta.annotation.PostConstruct;
import jakarta.annotation.PreDestroy;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Component;

@Component
public class Student {
```

```
    private int roll_no;
    private String name;

    @Autowired
    @Qualifier("homeAddress")  // Choosing specific Address Bean
    private Address address;

    public Student() {
        System.out.println(" 🔹 Constructor: Student Bean Created");
    }

    @PostConstruct
    public void init() {
        System.out.println("✅ @PostConstruct: Initializing Student Bean");
    }

    public void display() {
        System.out.println("Roll_no: " + roll_no);
        System.out.println("Name: " + name);
        System.out.println("Address: " + address);
    }

    @PreDestroy
    public void cleanup() {
        System.out.println("🗑️ @PreDestroy: Cleaning up Student Bean");
    }
}
```

2️⃣ Address.java (Base Interface)

```
java
CopyEdit
package in.sp.beans;
```

```
public interface Address {
    void setHouseno(int houseno);
    void setCity(String city);
    void setPincode(int pincode);
}
```

3️⃣ HomeAddress.java (One Implementation of Address)

```
java
CopyEdit
package in.sp.beans;

import jakarta.annotation.PostConstruct;
import jakarta.annotation.PreDestroy;
import org.springframework.stereotype.Component;

@Component("homeAddress")
public class HomeAddress implements Address {
    private int houseno;
    private String city;
    private int pincode;

    public HomeAddress() {
        System.out.println("🏡 Constructor: HomeAddress Bean Created");
    }

    @PostConstruct
    public void init() {
        System.out.println("✅ @PostConstruct: HomeAddress Bean Initialized");
    }

    @Override
    public void setHouseno(int houseno) { this.houseno = houseno; }
```

```
    @Override
    public void setCity(String city) { this.city = city; }

    @Override
    public void setPincode(int pincode) { this.pincode = pincode; }

    @Override
    public String toString() {
        return "#" + houseno + ", " + city + " -" + pincode;
    }

    @PreDestroy
    public void cleanup() {
        System.out.println("🗑️ @PreDestroy: Cleaning up HomeAddress Bean");
    }
}
```

4️⃣ OfficeAddress.java (Another Implementation of Address)

```
java
CopyEdit
package in.sp.beans;

import jakarta.annotation.PostConstruct;
import jakarta.annotation.PreDestroy;
import org.springframework.stereotype.Component;

@Component("officeAddress")
public class OfficeAddress implements Address {
    private int houseno;
    private String city;
    private int pincode;

    public OfficeAddress() {
```

```
        System.out.println("🏢 Constructor: OfficeAddress Bean Created");
    }

    @PostConstruct
    public void init() {
        System.out.println("✅ @PostConstruct: OfficeAddress Bean Initialized");
    }

    @Override
    public void setHouseno(int houseno) { this.houseno = houseno; }

    @Override
    public void setCity(String city) { this.city = city; }

    @Override
    public void setPincode(int pincode) { this.pincode = pincode; }

    @Override
    public String toString() {
        return "#" + houseno + ", " + city + " -" + pincode;
    }

    @PreDestroy
    public void cleanup() {
        System.out.println("🗑️ @PreDestroy: Cleaning up OfficeAddress Bean");
    }
}
```

🛠️ How Will This Work?

🔷 When the Spring Application Starts:

1️⃣ Spring creates beans

- `Student` bean is created → `Student()` constructor runs

- `HomeAddress` bean is created → `HomeAddress()` constructor runs
  2️⃣
  PostConstruct methods run
- `@PostConstruct` in `HomeAddress` → Initializes address
- `@PostConstruct` in `Student` → Initializes student

🔷 When the Application Shuts Down:

1️⃣ **PreDestroy methods run**

- `@PreDestroy` in `Student` → Cleans up Student bean
- `@PreDestroy` in `HomeAddress` → Cleans up address bean
  2️⃣
  Spring removes the beans

📌 Expected Console Output

```
graphql
CopyEdit
🏡 Constructor: HomeAddress Bean Created
✅ @PostConstruct: HomeAddress Bean Initialized
🔹 Constructor: Student Bean Created
✅ @PostConstruct: Initializing Student Bean

🗑️ @PreDestroy: Cleaning up Student Bean
🗑️ @PreDestroy: Cleaning up HomeAddress Bean
```

📌 Key Takeaways

✔️ `@PostConstruct` → Runs **after dependency injection**, useful for setup tasks.

✔️ `@PreDestroy` → Runs **before bean destruction**, useful for cleanup tasks.

✔️ Works for **singleton** beans but **not prototype beans**.

✔️ Used inside **any Spring-managed bean (service, repository, controller, etc.)**.

✅ What You CAN Do in Lifecycle Methods

✔️ **Method (with no arguments, return type void)**

✔️ **Used for initialization (`@PostConstruct`) or cleanup (`@PreDestroy`)**

✔️ **Can be in any Spring-managed bean (`@Component`, `@Service`, `@Repository`, etc.)**

✔️ **Can call other methods inside these lifecycle methods**

❌ What You CANNOT Do in Lifecycle Methods

❌ **Cannot use a Constructor instead of `@PostConstruct`**

❌ **Cannot pass parameters to the method**

❌ **Cannot return a value (must be `void`)**

❌ **Cannot use `@PreDestroy` in a Prototype Bean (because Spring does not manage its destruction)**

✅ Correct Usage (Works)

```
java
CopyEdit
@PostConstruct
public void init() {
    System.out.println("✅ @PostConstruct: Initializing Student");
    setupDatabaseConnection(); // Calling another method
}
```

```
java
CopyEdit
@PreDestroy
public void cleanup() {
    System.out.println("🗑️ @PreDestroy: Cleaning up Student");
```

```
    closeDatabaseConnection(); // Calling another method
}
```

❌ Wrong Usage (Won't Work)

❌ Cannot Use Parameters

```
java
CopyEdit
@PostConstruct
public void init(int id) {  // ❌ Not allowed, parameters are not supported
    System.out.println("Initializing with ID: " + id);
}
```

❌ Cannot Return a Value

```
java
CopyEdit
@PreDestroy
public String cleanup() {  // ❌ Not allowed, must be void
    return "Cleanup complete";
}
```

❌ Cannot Replace @PostConstruct with Constructor

```
java
CopyEdit
@Component
public class Student {
    public Student() {  // ❌ Constructor is NOT the same as @PostConstruct
        System.out.println("Student Constructor: Bean Created");
```

```
    }
  }
```

👉 **Use `@PostConstruct` instead for lifecycle-related setup!**

📌 Summary

✔️ ✅ Use `void` methods without arguments inside lifecycle annotations

✔️ ✅ Can call other methods inside `@PostConstruct` & `@PreDestroy`

✔️ ❌ Cannot use constructors, parameters, or return values

✔️ ❌ `@PreDestroy` does **not** work in **prototype-scoped** beans