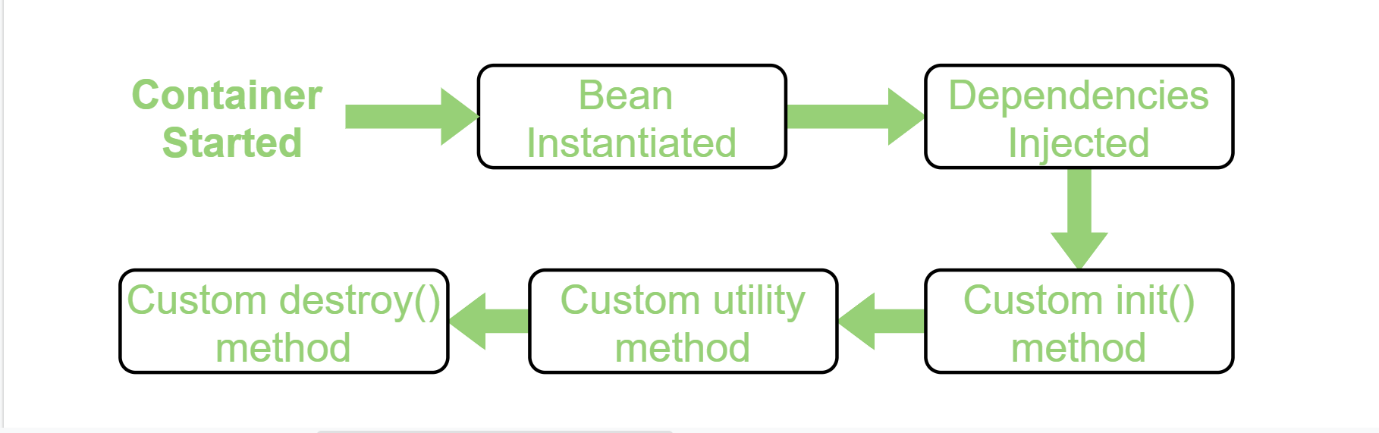
*The lifecycle of any object means when & how it is born, how it behaves throughout its life, and when & how it dies. Similarly, the bean life cycle refers to when & how the bean is instantiated, what action it performs until it lives, and when & how it is destroyed.*

*Bean life cycle is managed by the spring container. When we run the program then, first of all, the spring container gets started. After that, the container creates the instance of a bean as per the request, and then dependencies are injected. And finally, the bean is destroyed when the spring container is closed.*

*if we want to execute some code on the bean instantiation and just after closing the spring container, then we can write that code inside the custom* ***init()*** *method and the* ***destroy()*** *method.*



*Spring provides three ways to implement the life cycle of a bean.*

* *By XML: In this approach, in order to avail custom init() and destroy() methods for a bean we have to register these two methods inside the Spring XML configuration file while defining a bean.*

|  |
| --- |
| **package** beans;    **public** **class** HelloWorld {    **public** **void** init() **throws** Exception      {          System.out.println(              "Bean HelloWorld has been "              + "instantiated and I'm "              + "the init() method");      }    **public** **void** destroy() **throws** Exception      {          System.out.println(              "Container has been closed "              + "and I'm the destroy() method");      }  } |
| <**beans**>      <**bean** id="hw" class="beans.HelloWorld"              init-method="init" destroy-method="destroy"/>    </**beans**> |

* *By Programmatic Approach - To provide the facility to the created bean to invoke custom init() method on the startup of a spring container and to invoke the custom destroy() method on closing the container, we need to implement our bean with two interfaces namely InitializingBean, DisposableBean and will have to override afterPropertiesSet() and destroy() method. afterPropertiesSet() method is invoked as the container starts and the bean is instantiated whereas, the destroy() method is invoked just after the container is closed.*

**public** **class** HelloWorld

**implements** InitializingBean,

 DisposableBean {

    @Override

**public** **void** afterPropertiesSet()

**throws** Exception

    {

        System.out.println(

            "Bean HelloWorld has been "

            + "instantiated and I'm the "

            + "init() method");

    }

    @Override

**public** **void** destroy() **throws** Exception

    {

        System.out.println(

            "Container has been closed "

            + "and I'm the destroy() method");

    }

* *Using Annotation: To provide the facility to the created bean to invoke custom init() method on the startup of a spring container and to invoke the custom destroy() method on closing the container, we need to annotate init() method by @PostConstruct annotation and destroy() method by @PreDestroy annotation.*

*Spring Bean Life Cycle Stages –*

* *Instantiation: Setting the Foundation - The journey of a Spring bean begins with its instantiation. During this stage, the container creates a new instance of the bean by invoking its constructor. The primary purpose of this stage is to prepare the bean for further initialization.*

*Purpose: Initialize essential properties and resources.*

*Set up the initial state of the bean.*

*When to Use: Set initial values or configurations.*

*Example: Instantiation is like giving birth to a new character in a game.*

* *Population of Properties: Filling the Gaps - At this stage, the Spring container injects properties and dependencies into the bean.*

*Purpose: Inject properties and dependencies into the bean.*

*Configure the bean with the required resources.*

*When to Use: To populate bean properties from the context.*

*Example: Heroes arm themselves with weapons and tools.*

* *BeanNameAware: Giving Identity - After instantiation, the bean becomes aware of its assigned name within the Spring container*

*Purpose: Allow beans to know their assigned name.*

*Access the name assigned to the bean.*

*When to Use: When the bean needs to reference itself within the context.*

*Example: With the character created, let’s give it a name.*

* *BeanFactoryAware and ApplicationContextAware: Embracing Context - The next stage involves the bean gaining awareness of the bean factory or application context it’s part of.*

*Purpose: Access the broader application context.*

*Interact with other beans and resources.*

*When to Use: To access global context and resources.*

*Example: Our character doesn’t live in isolation; it’s part of a bigger world.*

* *BeanPostProcessor: Adding Magic - During this phase, custom logic can be executed before and after the bean’s initialization.*

*Purpose: Execute custom logic before and after initialization. Customize the bean’s behavior during creation.*

*When to Use: For adding custom behavior during bean creation.*

*Example: Our gaming character might need some magical enhancements, right? Similarly, during the BeanPostProcessor stage, you can sprinkle some magic on your beans.*

* *@PostConstruct: Customizing Bean Initialization - The @PostConstruct annotation plays a significant role in customizing the initialization of a bean.*

*Purpose: Execute custom logic after a bean has been initialized.*

*When to Use:*

*Customization: Utilize @PostConstruct when you want to add your own custom behavior to a bean after its creation.*

*Final Setup: It’s useful when you need to perform specific actions on the bean right after it’s initialized, such as configuring properties or performing last-minute preparations.*

*Example: Our gaming character might need to perform some readiness checks before it can enter the virtual world.*

* *InitializingBean: Preparing for Action - As the bean gets closer to being fully operational, it enters the initialization stage. This stage provides an opportunity for executing custom setup logic after properties are set and before the bean is ready for use.*

*Purpose: Perform custom setup and initialization logic.*

*Execute actions after properties are set.*

*When to Use: When additional setup beyond the constructor is needed.*

*Example: After character creation and magical enchantments, it’s time for the character to gear up and prepare for its adventures.*

* *Custom Initialization: Tailored Setup - In some scenarios, the need arises for specialized initialization steps that go beyond standard setup. This is where custom initialization methods come into play.*

*Purpose: Perform unique setup tasks for the bean.*

*Execute specialized initialization logic.*

*When to Use: When specific initialization is required.*

*Example: Heroes have unique abilities that require special training. Similarly, beans can undergo personalized preparation during this stage.*

* *@PreDestroy: Preparing for Cleanup - The @PreDestroy annotation marks a method within a bean as a pre-destruction callback. It allows to perform cleanup operations and release resources gracefully before the bean’s final disposal.*

*Purpose: Implement custom cleanup operations before the destruction of the bean.*

*Ensure proper resource release and perform any necessary cleanup tasks.*

*When to Use: When you need to execute specific cleanup logic just before a bean is destroyed.*

*To ensure that resources, such as open files or database connections, are properly released.*

*Example: In our gaming world, every character has a special ritual they perform just before they bid farewell and prepare to rest.*

* *DisposableBean: Bidding Farewell - As beans live their lifecycle, they eventually face the end. The DisposableBean interface provides a method for executing cleanup operations before the bean is destroyed.*

*Purpose: Implement cleanup operations before destruction.*

*Ensure proper resource release and cleanup.*

*When to Use: To perform cleanup actions before destruction.*

*Example: Every adventure has its conclusion, and so does the life cycle of a bean.*

* *Custom Destruction: Parting Moments - In addition to standard destruction methods, you can implement custom destruction logic.*

*Purpose: Execute custom logic before destruction.*

*Perform specialized cleanup actions.*

*When to Use: When unique cleanup is necessary.*

*Example: The customDestroy method is a custom-defined routine that a character, represented by this method,*