Bean Lifecycle

Spring provides various mechanisms for getting bean lifecycle callbacks. These callbacks are useful to take some specific action at a particular lifecycle stage. It's specially needed at the point when bean is fully initialized and all properties are set.

Following are the differents ways to receive callbacks after bean has initialized and before bean is destroyed:

- Using @PostConstruct and @PreDestroy
- Using 'initMethod' and 'destroyMethod' of @Bean annotation.
- By implementing InitializingBean and DisposableBean

1 - Receiving Lifecycle callbacks by using @PostConstruct and @PreDestroy

The recommended way to receive initialization/destruction callbacks is by using @PostConstruct and @PreDestroy annotations

```
public class OtherBean {
  private String message;

  public OtherBean(String message) {
     this.message = message;
  }

  @Override
```

```
  public String toString() {
      return "OtherBean{" +
              "message='" + message + '\'' +
              '}';
  }
}

public class MyBean {
  private OtherBean otherBean;

  public MyBean() {
      System.out.println("MyBean constructor");
  }

  @PostConstruct
  public void myPostConstruct() {
      System.out.println(" @PostConstruct method");
  }

  @Autowired
  public void setOtherBean(OtherBean otherBean) {
      System.out.println("setOtherBean(): " + otherBean);
      this.otherBean = otherBean;
```

```
  }

  public void doSomething() {
      System.out.println("doSomething()");
  }

  @PreDestroy
  public void cleanUp() {
      System.out.println("@PreDestroy method");
  }
}
```

In above example, we used setter injection, we can instead use field/constructor injection as well, the fields will be fully initialized the time when @PostConstruct method is called.

```
@Configuration
public class LifeCycleExample {

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

  @Bean
```

```
  public OtherBean otherBean() {
      return new OtherBean("hello from otherBean!");
  }

  public static void main(String[] args) {
      ConfigurableApplicationContext context =
              new AnnotationConfigApplicationContext(LifeCycleExample.class);

      context.registerShutdownHook();

      System.out.println("-- accessing bean --");
      MyBean bean = context.getBean(MyBean.class);
      bean.doSomething();

      System.out.println("-- finished --");
  }
}
```

Output

```
MyBean constructor
setOtherBean(): OtherBean{message='hello from otherBean!'}
 @PostConstruct method
-- accessing bean --
doSomething()
```

-- finished --

@PreDestroy method

Understanding registerShutdownHook() method

In above example, we used ConfigurableApplicationContext#registerShutdownHook(). This method registers a shutdown hook with the JVM runtime. This hook receives notification on JVM shutdown, at that time it closes the underlying context and calls all @PreDestroy and other standard register destroy methods. If we don't want to use this method then we have to call ConfigurableApplicationContext#close() ourselves at JVM shutdown, otherwise our destroy methods won't get called.

2 - Receiving lifecycle callbacks by using 'initMethod' and 'destroyMethod' of @Bean annotation

Spring's @Bean annotation provides following optional elements to received lifecycle callbacks:

initMethod: specifies the method name to received callback after bean initialization.

destroyMethod: specified the method name to receive callback before bean destruction.

Example

```
public class OtherBean {
  private String message;

  public OtherBean(String message) {
      this.message = message;
```

```
  }

  @Override
  public String toString() {
      return "OtherBean{" +
              "message='" + message + '\'' +
              '}';
  }
}

public class MyBean {
  private OtherBean otherBean;

  public MyBean() {
      System.out.println("MyBean constructor");
  }

  public void myPostConstruct() {
      System.out.println("myPostConstruct()");
  }

  @Autowired
  public void setOtherBean(OtherBean otherBean) {
```

```
        System.out.println("setOtherBean(): " + otherBean);
        this.otherBean = otherBean;
    }

    public void doSomething() {
        System.out.println("doSomething()");
    }


    public void cleanUp() {
        System.out.println("cleanUp method");
    }
}


@Configuration
public class LifeCycleExample {

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

```
  @Bean
  public OtherBean otherBean() {
      return new OtherBean("Hello from otherBean!");
  }

  public static void main(String[] args) {
      ConfigurableApplicationContext context =
              new AnnotationConfigApplicationContext(LifeCycleExample.class);

      context.registerShutdownHook();

      System.out.println("-- accessing bean --");
      MyBean bean = context.getBean(MyBean.class);
      bean.doSomething();

      System.out.println("-- finished --");
  }
}
```

Output

```
MyBean constructor
setOtherBean(): OtherBean{message='Hello from otherBean!'}
myPostConstruct()
-- accessing bean --
```

```
doSomething()
-- finished --
cleanUp method
```

3 - Receiving lifecycle callbacks by implementing InitializingBean and DisposableBean

The interface InitializingBean has one method afterPropertiesSet which is called by spring framework after it has set all bean properties.

The interface DisposableBean has one method destroy which is called by spring framework when JVM sends the shutdown signal.

Example

```
public class OtherBean {
  private String message;

  public OtherBean(String message) {
      this.message = message;
  }

  @Override
  public String toString() {
      return "OtherBean{" +
```

```
            "message='" + message + '\'' +
            '}';
  }
}

public class MyBean implements InitializingBean, DisposableBean {
  private OtherBean otherBean;

  public MyBean() {
      System.out.println("MyBean constructor");
  }

  @Override
  public void afterPropertiesSet() {
      System.out.println("myPostConstruct()");
  }

  @Autowired
  public void setOtherBean(OtherBean otherBean) {
      System.out.println("setOtherBean(): " + otherBean);
      this.otherBean = otherBean;
  }
```

```
  public void doSomething() {
      System.out.println("doSomething()");
  }

  @Override
  public void destroy() {
      System.out.println("cleanUp method");
  }
}

@Configuration
public class LifeCycleExample {

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

  @Bean
  public OtherBean otherBean() {
      return new OtherBean("Hello from other Bean");
  }
```

```
  public static void main(String[] args) {
      ConfigurableApplicationContext context =
              new AnnotationConfigApplicationContext(LifeCycleExample.class);

      context.registerShutdownHook();

      System.out.println("-- accessing bean --");
      MyBean bean = context.getBean(MyBean.class);
      bean.doSomething();

      System.out.println("-- finished --");
  }
}
```

Output

```
MyBean constructor
setOtherBean(): OtherBean{message='Hello from other Bean'}
myPostConstruct()
-- accessing bean --
doSomething()
-- finished --
cleanUp method
```