Instructor Notes:

Add instructor notes here.

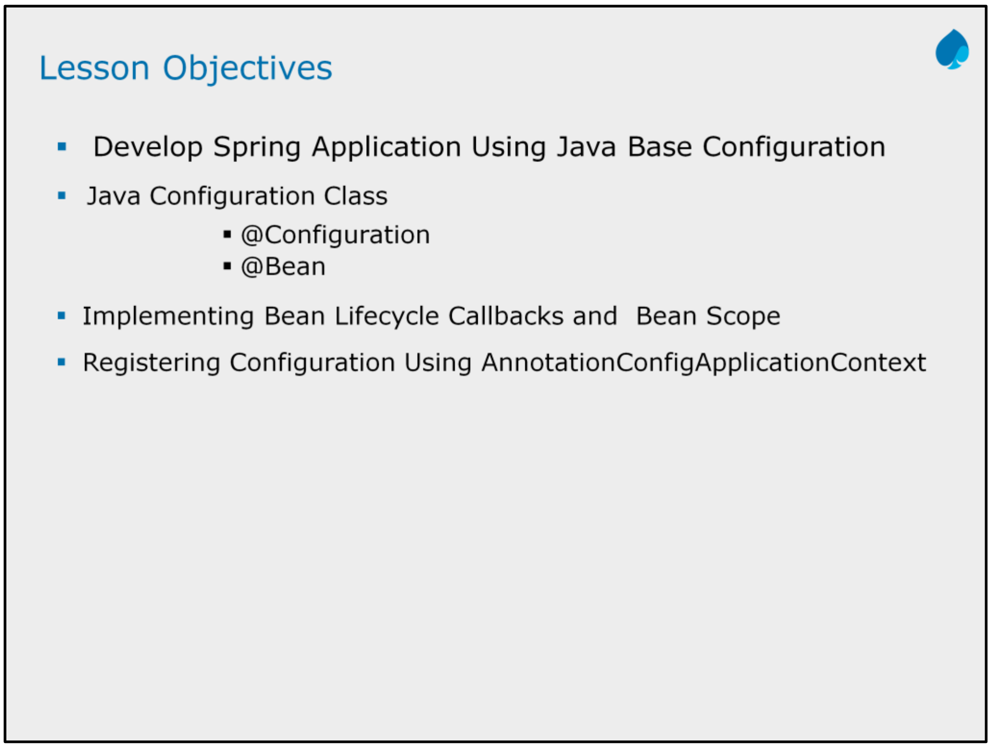
Lesson 3: Spring Java Base Configuration

Basic Spring 4.0

Capgemini

Instructor Notes:

Explain the lesson coverage

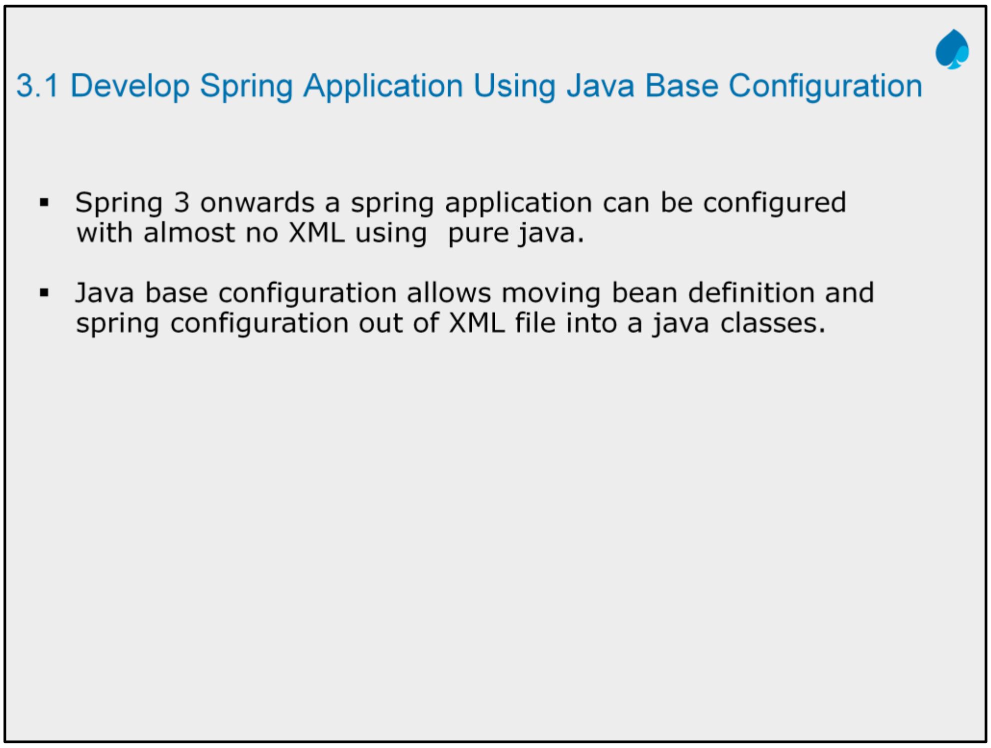


Lesson Objectives

- Develop Spring Application Using Java Base Configuration
- Java Configuration Class
  - @Configuration
  - @Bean
- Implementing Bean Lifecycle Callbacks and Bean Scope
- Registering Configuration Using AnnotationConfigApplicationContext

Instructor Notes:

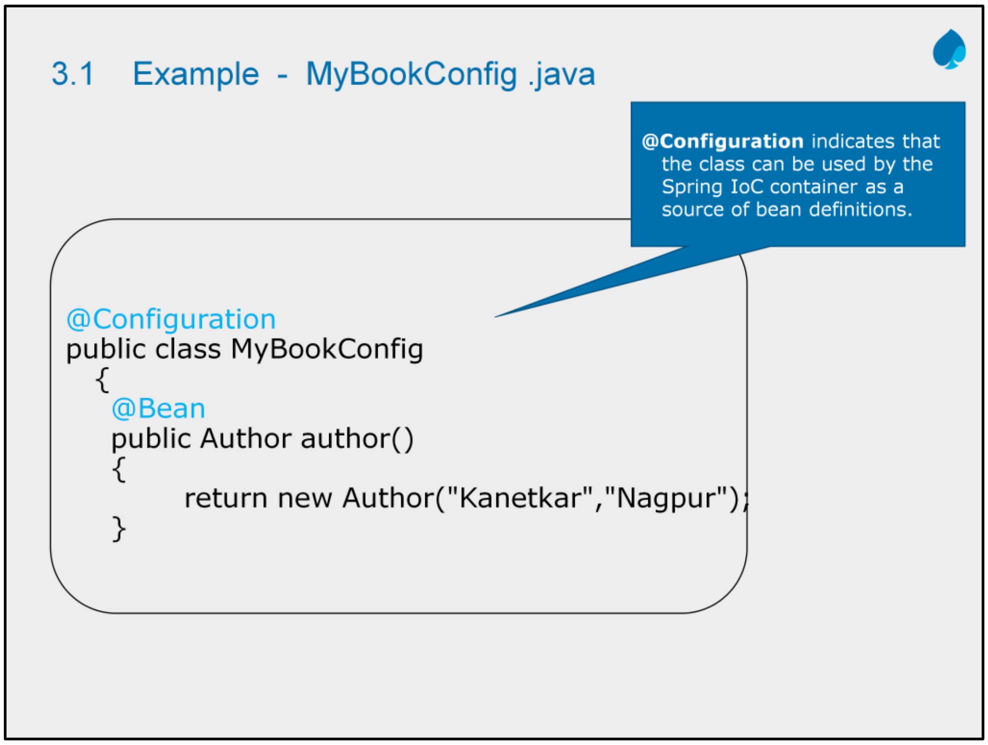
While there are several other Java expression languages available, OGNL, MVEL, and JBoss EL, to name a few, the Spring Expression Language was created to provide the Spring community with a single well supported expression language that can be used across all the products in the Spring portfolio.



3.1 Develop Spring Application Using Java Base Configuration

- Spring 3 onwards a spring application can be configured with almost no XML using pure java.
- Java base configuration allows moving bean definition and spring configuration out of XML file into a java classes.

Instructor Notes:



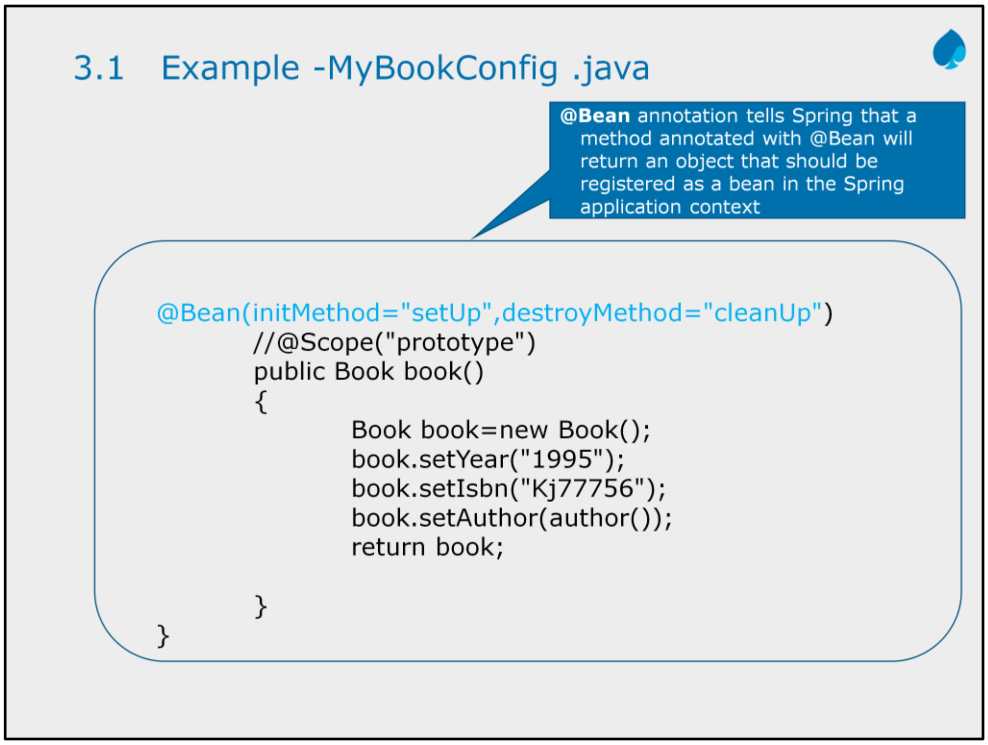
3.1 Example - MyBookConfig .java

@Configuration indicates that the class can be used by the Spring IoC container as a source of bean definitions.

```
@Configuration
public class MyBookConfig
   {
     @Bean
     public Author author()
     {
            return new Author("Kanetkar","Nagpur");
     }
```

Annotating a class with the **@Configuration** indicates that the class can be used by the Spring IoC container as a source of bean definitions.

Instructor Notes:



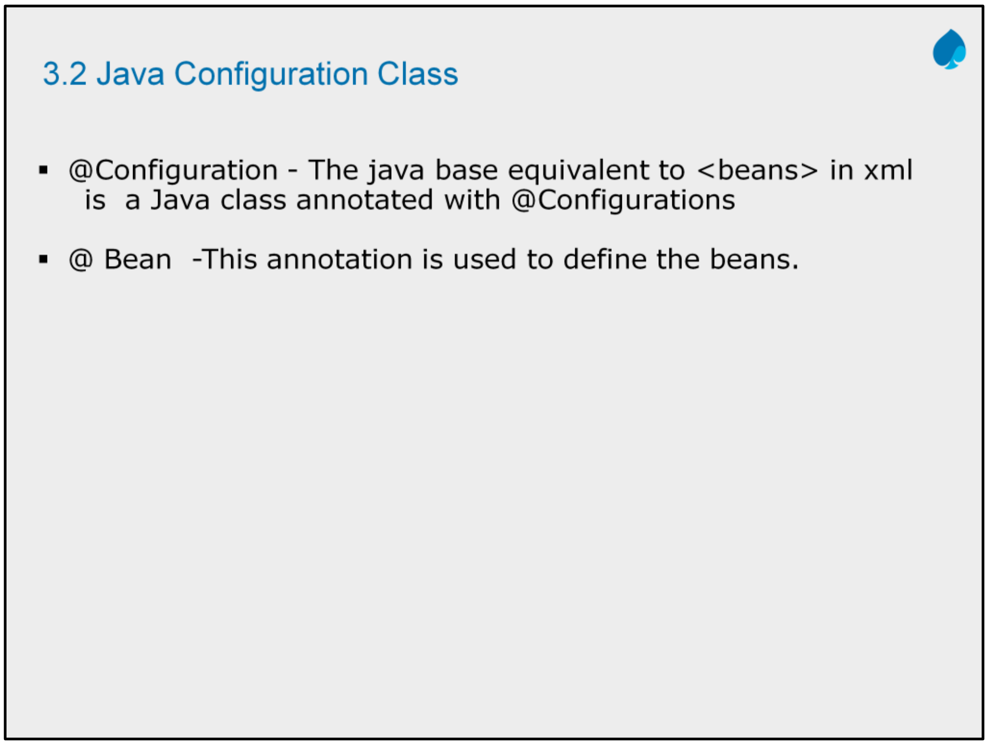
3.1 Example -MyBookConfig .java

@Bean annotation tells Spring that a method annotated with @Bean will return an object that should be registered as a bean in the Spring application context

```
@Bean(initMethod="setUp",destroyMethod="cleanUp")
        //@Scope("prototype")
        public Book book()
        {
                Book book=new Book();
                book.setYear("1995");
                book.setIsbn("Kj77756");
                book.setAuthor(author());
                return book;

        }
}
```

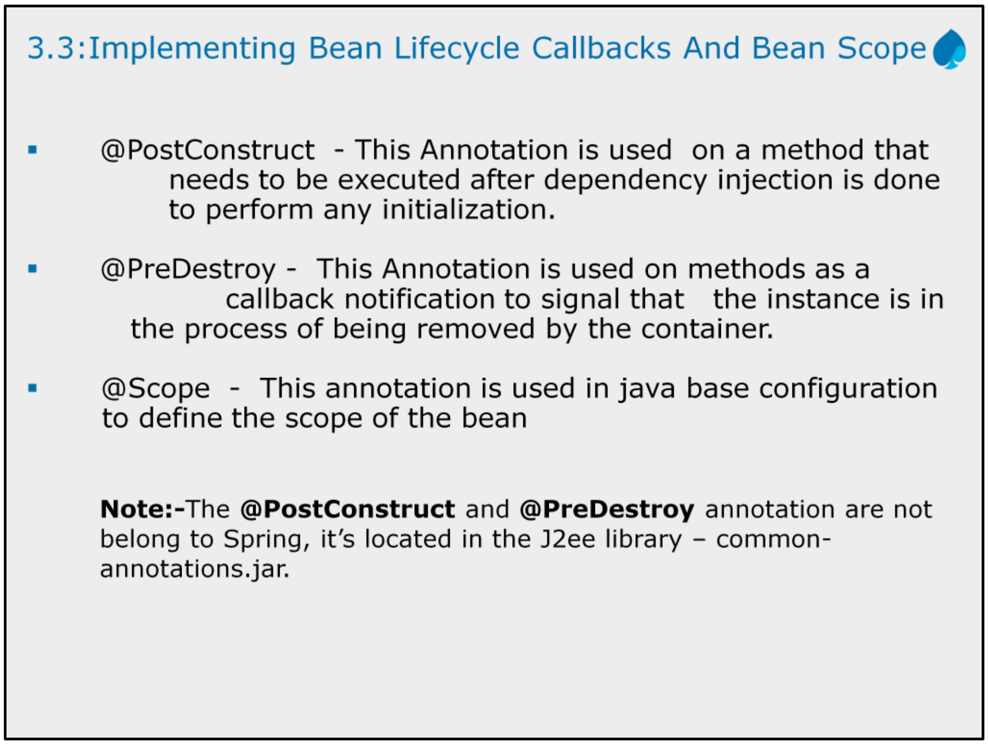
Instructor Notes:



3.2 Java Configuration Class

- @Configuration - The java base equivalent to <beans> in xml is a Java class annotated with @Configurations
- @ Bean -This annotation is used to define the beans.

Instructor Notes:

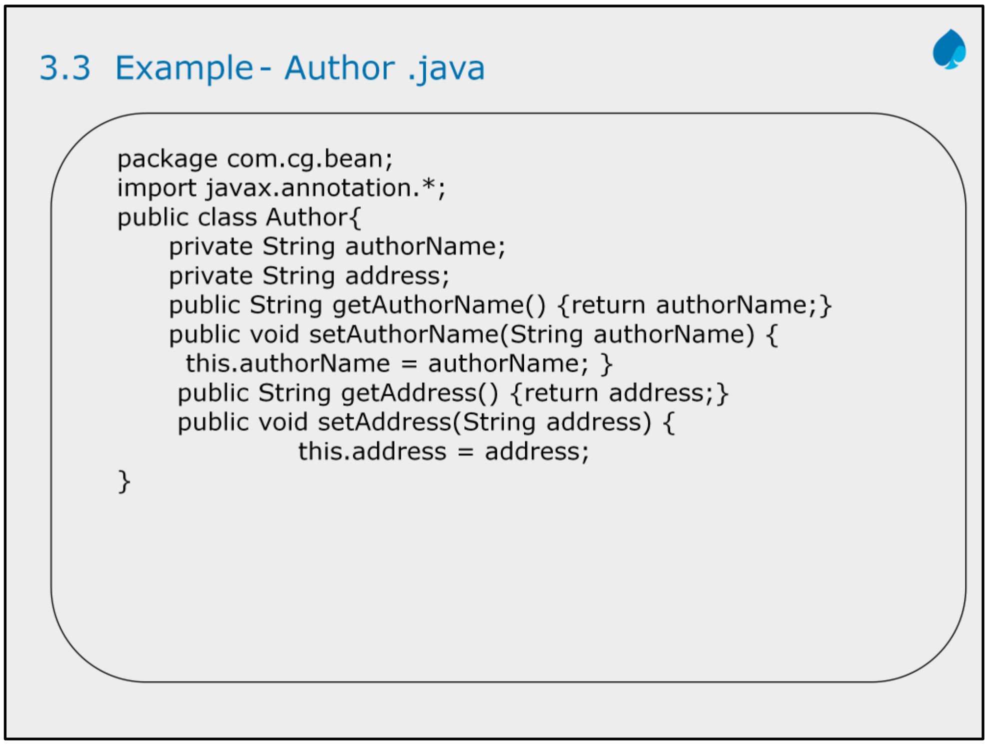


3.3:Implementing Bean Lifecycle Callbacks And Bean Scope

- @PostConstruct - This Annotation is used on a method that needs to be executed after dependency injection is done to perform any initialization.
- @PreDestroy - This Annotation is used on methods as a callback notification to signal that the instance is in the process of being removed by the container.
- @Scope - This annotation is used in java base configuration to define the scope of the bean

Note:-The **@PostConstruct** and **@PreDestroy** annotation are not belong to Spring, it’s located in the J2ee library – common-annotations.jar.

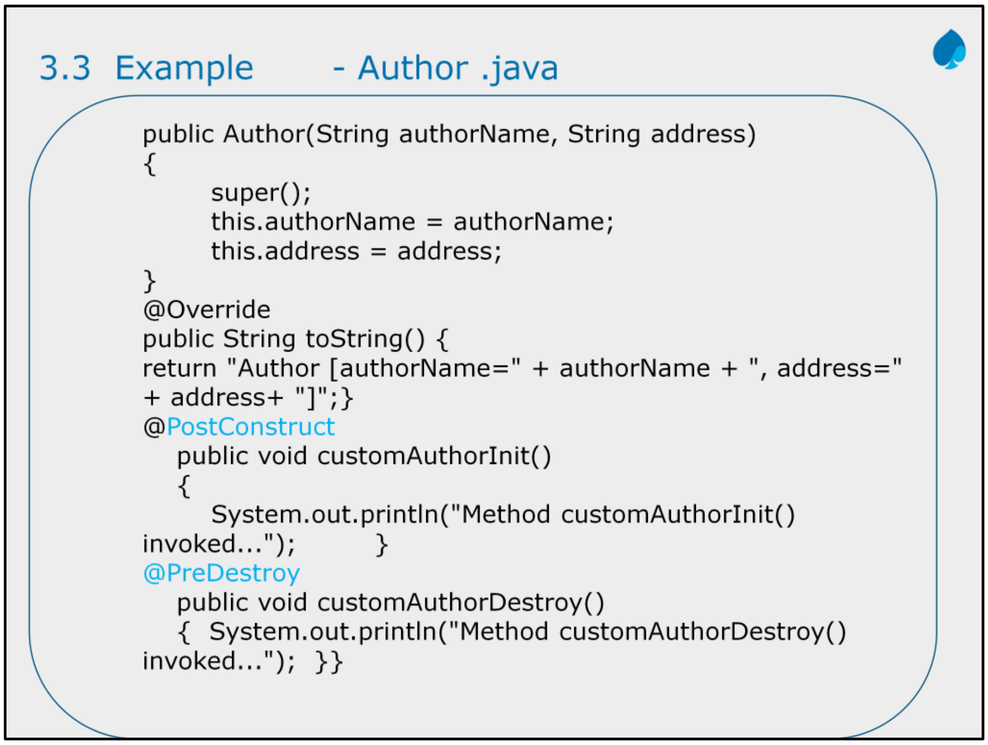
Instructor Notes:



3.3 Example - Author .java

```
package com.cg.bean;
import javax.annotation.*;
public class Author{
      private String authorName;
      private String address;
      public String getAuthorName() {return authorName;}
      public void setAuthorName(String authorName) {
        this.authorName = authorName; }
       public String getAddress() {return address;}
       public void setAddress(String address) {
                     this.address = address;
}
```

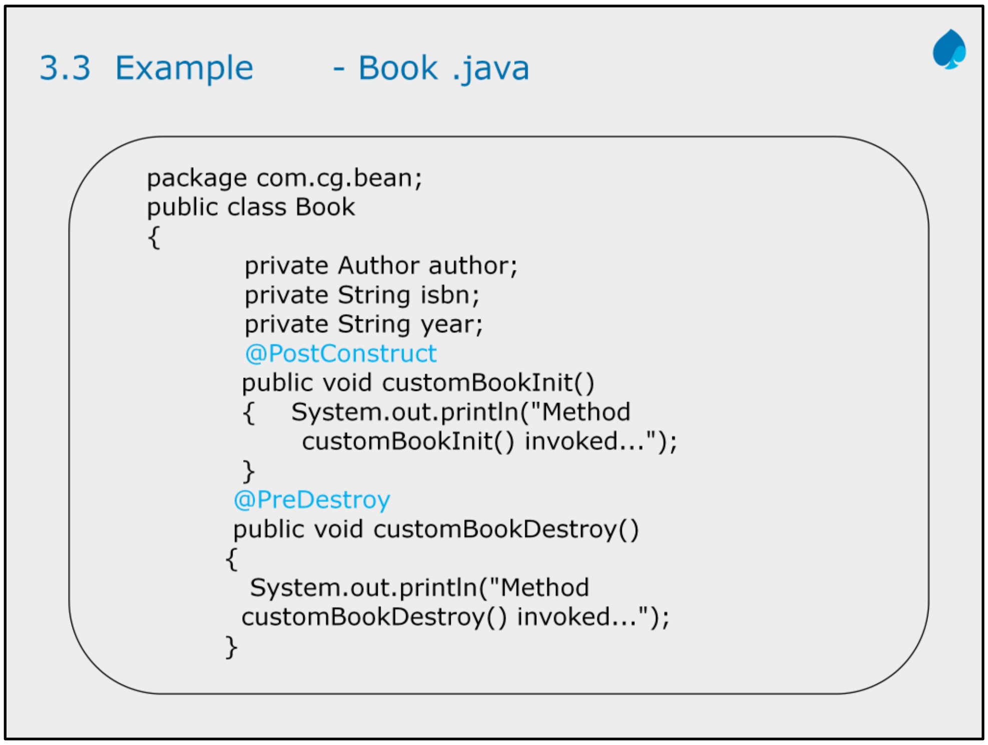
Instructor Notes:



3.3 Example - Author .java

```
public Author(String authorName, String address)
{
        super();
        this.authorName = authorName;
        this.address = address;
}
@Override
public String toString() {
return "Author [authorName=" + authorName + ", address="
+ address+ "]";}
@PostConstruct
    public void customAuthorInit()
    {
        System.out.println("Method customAuthorInit()
invoked...");         }
@PreDestroy
    public void customAuthorDestroy()
    {  System.out.println("Method customAuthorDestroy()
invoked...");  }}
```

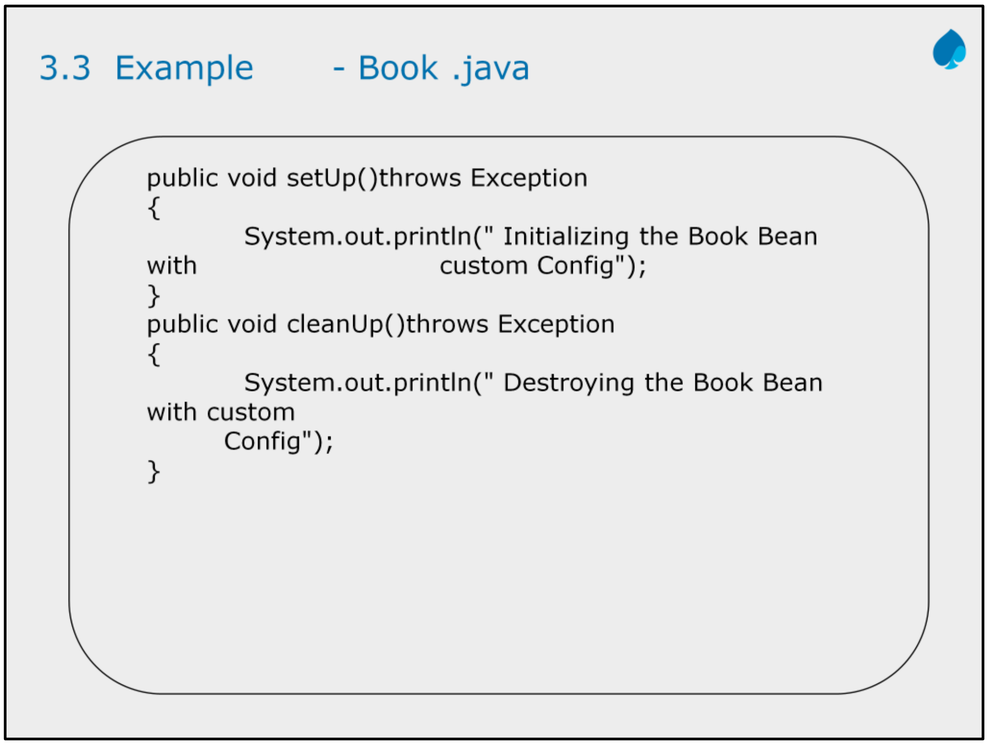
Instructor Notes:



3.3 Example - Book .java

```
package com.cg.bean;
public class Book
{
            private Author author;
            private String isbn;
            private String year;
            @PostConstruct
           public void customBookInit()
           {     System.out.println("Method
                  customBookInit() invoked...");
           }
          @PreDestroy
          public void customBookDestroy()
         {
            System.out.println("Method
           customBookDestroy() invoked...");
         }
```

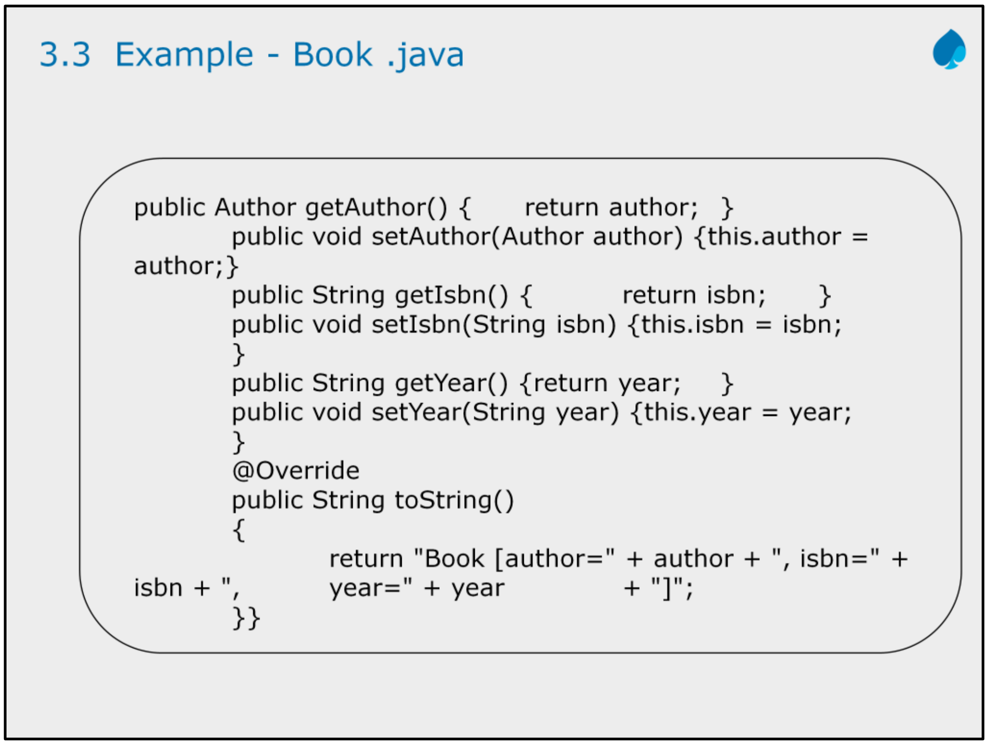
Instructor Notes:



3.3 Example - Book .java

```
public void setUp()throws Exception
{
        System.out.println(" Initializing the Book Bean
with                     custom Config");
}
public void cleanUp()throws Exception
{
        System.out.println(" Destroying the Book Bean
with custom
      Config");
}
```

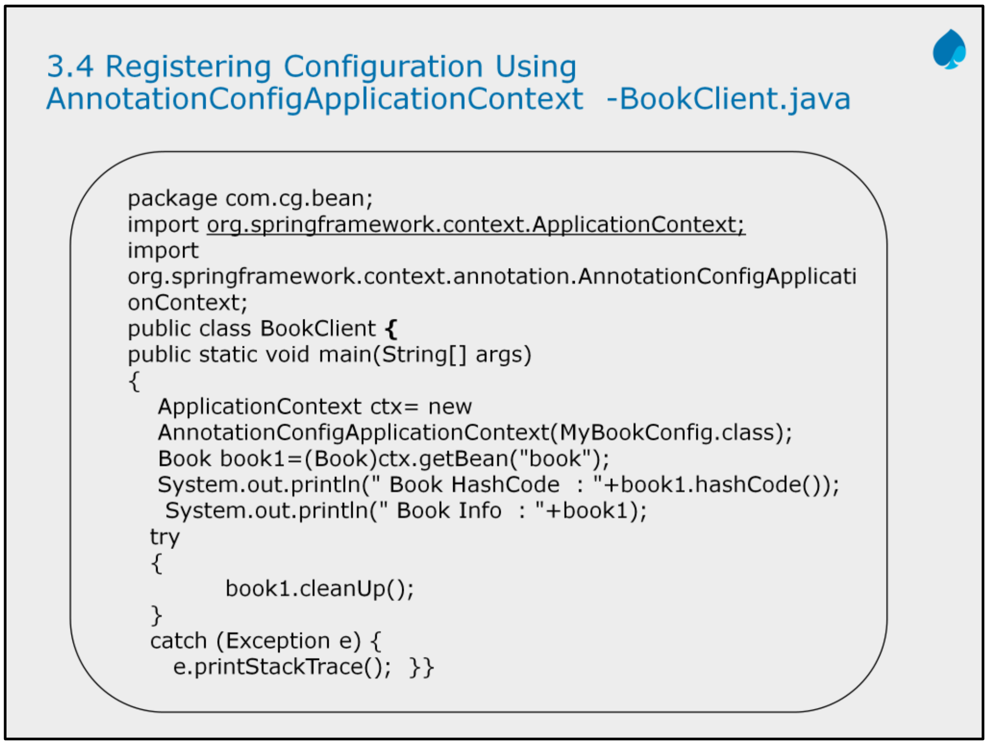
Instructor Notes:



3.3 Example - Book .java

```
public Author getAuthor() {      return author;   }
          public void setAuthor(Author author) {this.author =
author;}
          public String getIsbn() {          return isbn;      }
          public void setIsbn(String isbn) {this.isbn = isbn;
          }
          public String getYear() {return year;     }
          public void setYear(String year) {this.year = year;
          }
          @Override
          public String toString()
          {
                    return "Book [author=" + author + ", isbn=" +
isbn + ",          year=" + year              + "]";
          }}
```

Instructor Notes:



3.4 Registering Configuration Using AnnotationConfigApplicationContext -BookClient.java

```
package com.cg.bean;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApplicati
onContext;
public class BookClient {
public static void main(String[] args)
{
   ApplicationContext ctx= new
   AnnotationConfigApplicationContext(MyBookConfig.class);
   Book book1=(Book)ctx.getBean("book");
   System.out.println(" Book HashCode  : "+book1.hashCode());
    System.out.println(" Book Info  : "+book1);
  try
  {
          book1.cleanUp();
  }
  catch (Exception e) {
     e.printStackTrace();  }}
```

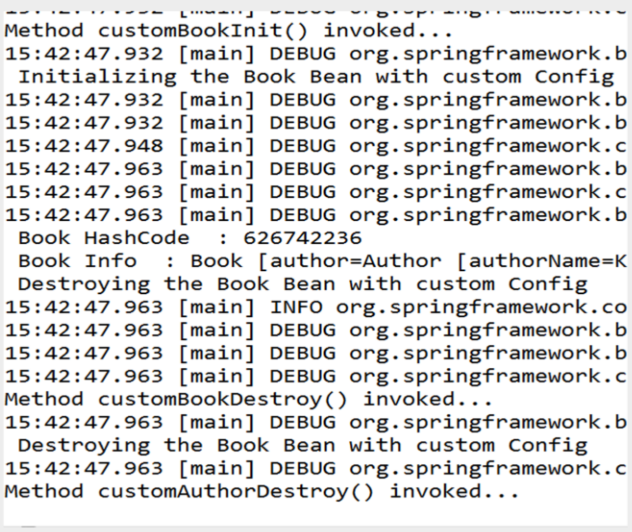
Bean in spring can be created in java file instead of XML,too. They are created with the help of @Configuration. And that application configuration is loaded by AnnotationConfigApplicationContext in spring container.

- ApplicationContext ctx= new AnnotationConfigApplicationContext(MyBookConfig.class);
  Create a new AnnotationConfigApplicationContext, deriving bean definitions from the given annotated classes and automatically refreshing the context.

- AnnotationConfigApplicationContext has the *register* method as well that accepts the bean configuration class.
- AnnotationConfigApplicationContext provides the method *getBean* to get the bean object.

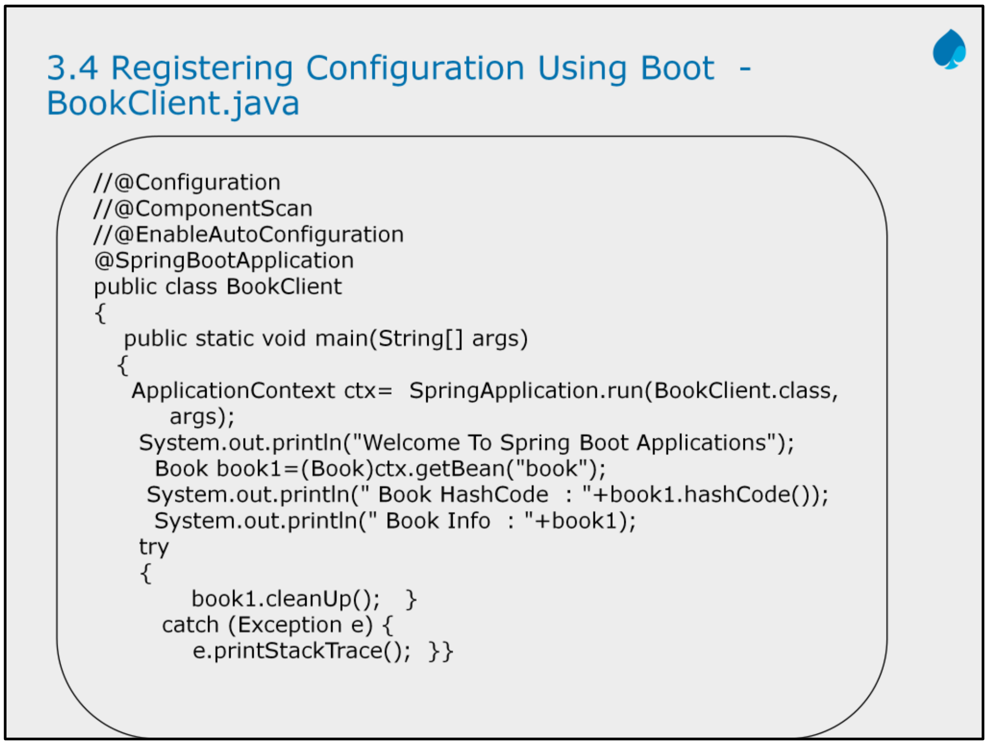
Instructor Notes:

3.4 Registering Configuration Using AnnotationConfigApplicationContext -BookClient.java



```
15:42:47.932 [main] DEBUG org.springframework.c
Method customBookInit() invoked...
15:42:47.932 [main] DEBUG org.springframework.b
 Initializing the Book Bean with custom Config
15:42:47.932 [main] DEBUG org.springframework.b
15:42:47.932 [main] DEBUG org.springframework.b
15:42:47.948 [main] DEBUG org.springframework.c
15:42:47.963 [main] DEBUG org.springframework.b
15:42:47.963 [main] DEBUG org.springframework.c
15:42:47.963 [main] DEBUG org.springframework.b
 Book HashCode   : 626742236
 Book Info   : Book [author=Author [authorName=K
 Destroying the Book Bean with custom Config
15:42:47.963 [main] INFO org.springframework.co
15:42:47.963 [main] DEBUG org.springframework.b
15:42:47.963 [main] DEBUG org.springframework.b
15:42:47.963 [main] DEBUG org.springframework.c
Method customBookDestroy() invoked...
15:42:47.963 [main] DEBUG org.springframework.b
 Destroying the Book Bean with custom Config
15:42:47.963 [main] DEBUG org.springframework.c
Method customAuthorDestroy() invoked...
```

Instructor Notes:



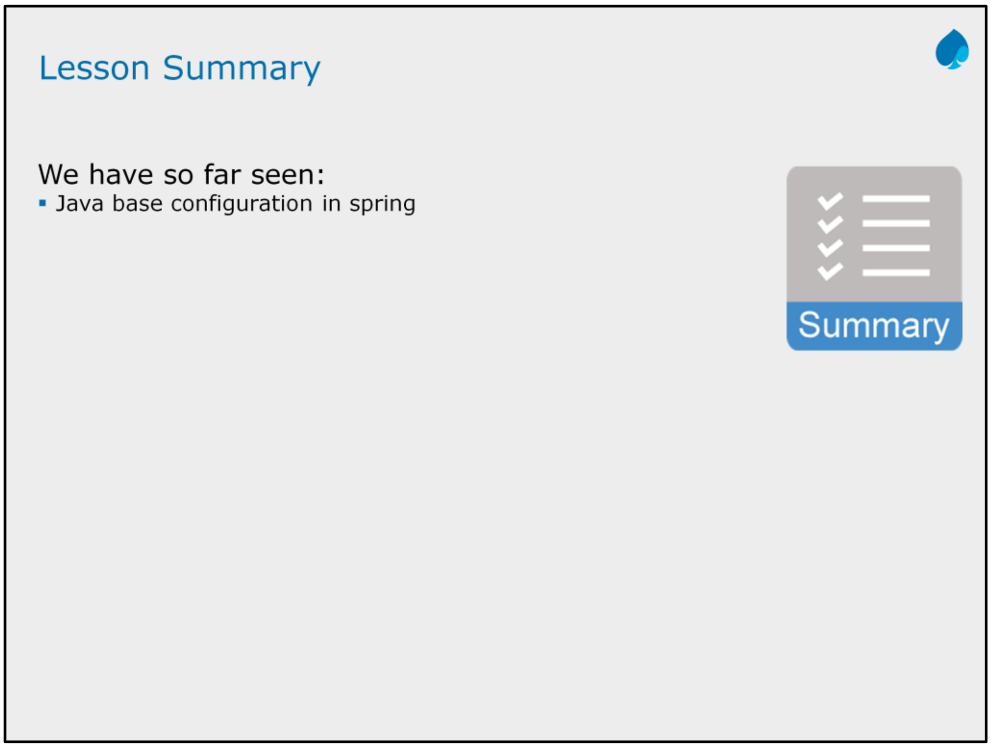
3.4 Registering Configuration Using Boot - BookClient.java

```
//@Configuration
//@ComponentScan
//@EnableAutoConfiguration
@SpringBootApplication
public class BookClient
{
   public static void main(String[] args)
  {
    ApplicationContext ctx=  SpringApplication.run(BookClient.class,
        args);
     System.out.println("Welcome To Spring Boot Applications");
       Book book1=(Book)ctx.getBean("book");
      System.out.println(" Book HashCode  : "+book1.hashCode());
       System.out.println(" Book Info  : "+book1);
     try
     {
          book1.cleanUp();   }
       catch (Exception e) {
          e.printStackTrace();  }}
```

1) @EnableAutoConfiguration annotation auto-configures the beans that are present in the classpath. This simplifies the developers work by guessing the required beans from the classpath and configure it to run the application. This annotation is part of the spring boot project.
2) With the spring boot 1.2.0 release, the need for this annotation has been reduced because there is an alternative annotation @SpringBootApplication which combines the three annotations @Configuration,@EnableAutoConfiguration and @ComponentScan.

Instructor Notes:

Additional notes for instructor



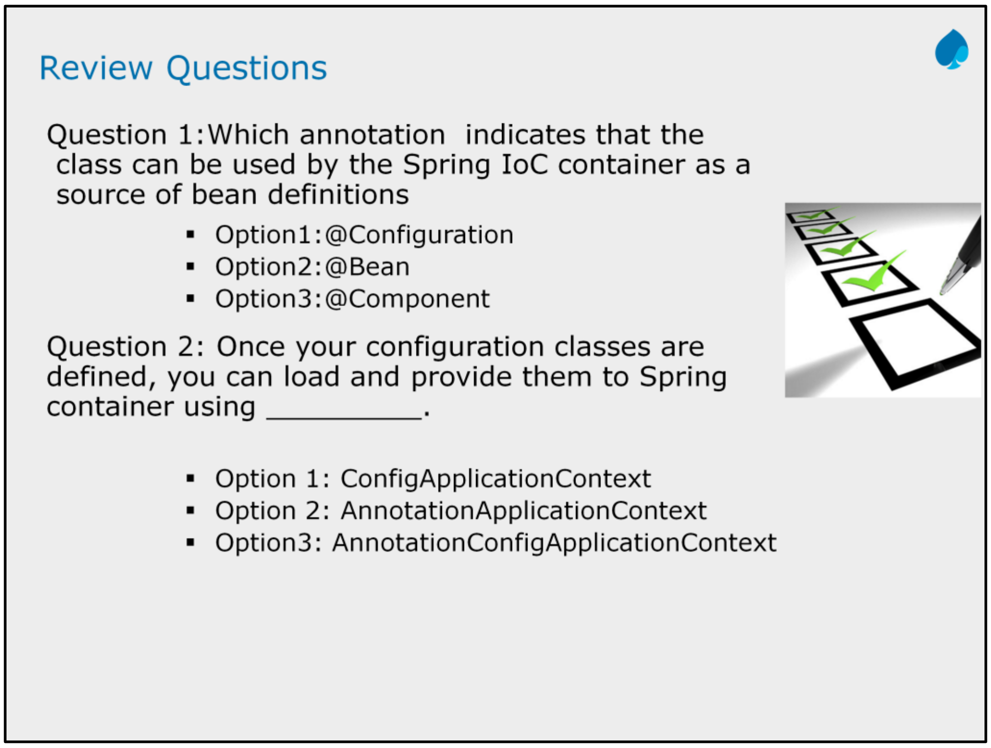
Lesson Summary

We have so far seen:

- Java base configuration in spring

Instructor Notes:

Ans-1 : Option 1
Ans-2 : Option 3



Review Questions

Question 1:Which annotation indicates that the class can be used by the Spring IoC container as a source of bean definitions

- Option1:@Configuration
- Option2:@Bean
- Option3:@Component

Question 2: Once your configuration classes are defined, you can load and provide them to Spring container using __________.

- Option 1: ConfigApplicationContext
- Option 2: AnnotationApplicationContext
- Option3: AnnotationConfigApplicationContext