List of Annotations used
========================
a. @Component
b. @Service
c. @Repository
d. @Primary
e. @Lazy
f. @Value
g. @PropertySource
h. @Autowired
i. @Qualifier
j. @Required
k. @Scope
l. @Controller

xml
==
<context:component-scan basepackage=""/>
<context:property-placeholder location=''/>
<context:annotation-config/>

SpringBean LifeCycle
====================
  1. Java class Life cycle
            a. static block
            b. instance block
            c. constructor
            d. setter
            e. using the created object,make a call to methods and execute Buisness logic.
            f. Destroy the Object.

2. Spring bean life cycle
                  Start the container****
            a. static block
            b. object instantiation
            c. custom init method(@PostConstruct) if successful then step d,e will be executed or else it won't execute.
            d. Business logic method
            e. custom destroy method(@PreDestroy)
                  Stop the container*****

                  refer:: IOCProject-27-SpringBeanLifeCycleActionApp

@PostConstruct and @PreDestroy annotations are supplied from jdk, so these annotations are deprecated in jdk9 and above version, so
to use these annoations in spring framework we need to add a special jar as shown below.
            https://mvnrepository.com/artifact/javax.annotation/javax.annotation-api/1.3.2

Advantages
==========
1. Makes spring bean as non-invasive.[working to a company without bond]
2. no need to configure spring bean life cycle methods anywhere, based on annotation automatically they will be executed.

DisAdvantage
============

1. We cannot use this approach for third party supplied pre-defined classes.

100%Code driven SpringApp development/Java Config Approach of SpringApp development
==========
 Advantages
      a. XMLBased cfg can be avoided in maximum cases
      b. Improves the readability
      c. Debugging becomes easy
      d. Foundation to learn SpringBoot

ThumbRule
=========
1. Configure userDefined classes as Springbean using Stereotype annotations(@Component) and link them with Configuration class
   alternative to SpringBean cfg file(xml file) using @ComponentScan
           note: Java class that is annotated with @Configuration automatically becomes Configuration class

2. Configure PreDefined class as Spring beans using @Bean methods(method that is annotated with @Bean) of @Configuration class.

3. use AnnotationConfigApplicationContext class to create an IOC container havaing @Configuration class as the input classname

Note: @Configuration class is internally a Spring bean becoz @Configuration internally contains @Component.

AppConfig.java(Alternative to XML)
==========
```
package in.ineuron.cfg;

import java.time.LocalDateTime;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan(basePackages = "in.ineuron")
public class AppConfig {

	static {
		System.out.println("AppConfig.class file is loading...");
	}

	public AppConfig() {
		System.out.println("AppConfig object is created:: Zero param constructor...");
	}

	@Bean(name = "dt")
	public LocalDateTime getSysDateTime() {
		System.out.println("AppConfig.getSysDateTime()");
		LocalDateTime date = LocalDateTime.now();
		return date;
	}
}
```

Test.java
=========

```
package in.ineuron.main;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.support.AbstractApplicationContext;

import in.ineuron.cfg.AppConfig;
import in.ineuron.comp.WishMessageGenerator;

public class TestApp {

	public static void main(String[] args)throws Exception {

		ApplicationContext factory = new
AnnotationConfigApplicationContext(AppConfig.class);
		System.out.println("*****Container started*******\n");

		WishMessageGenerator wmg = factory.getBean(WishMessageGenerator.class);
		System.out.println(wmg);

		String msg = wmg.greetMessage("kohli");
		System.out.println(msg);

		((AbstractApplicationContext) factory).close();
		System.out.println("\n*****Container closed*******");

	}

}
```

Test.java
=========

```
package in.ineuron.main;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.support.AbstractApplicationContext;

import in.ineuron.cfg.AppConfig;
import in.ineuron.comp.WishMessageGenerator;

public class TestApp {

	public static void main(String[] args)throws Exception {

		ApplicationContext factory = new
AnnotationConfigApplicationContext(AppConfig.class);
		System.out.println("*****Container started*******\n");

		WishMessageGenerator wmg = factory.getBean(WishMessageGenerator.class);
		System.out.println(wmg);

		String msg = wmg.greetMessage("kohli");
		System.out.println(msg);

		((AbstractApplicationContext) factory).close();
		System.out.println("\n*****Container closed*******");
	}
```

```
}
```

Usage of @Import(In 100%purejava code configuration)
==========

```
@Import(Resource = {Persistence.class,Service.class,Controller.class})
@Configuration
public class AppConfig{

}
```

How to load data from properties file into SpringBean?
     a. Using application.properties file ====> @PropertyResource(value="location of properties file") and binding it to @Value("${}") of field
      b. Using application.properties file ====> Environment(I) =====> Inside bean call environment.getProperty("")

refer:: IOCProject-29-RealTimeDI-100%SpringAnnotationsCRUDAPP