SpringApp can be developed in 4 mode
a. XML Configuration
b. XML + Annotation configuration
c. Pure Java Configuaration(No XML)
d. Auto Driven Configuration(SpringBoot ===> Ready made configurations are supplied)

AnnotationConfiguration
=======================
1.@Component
2.@Required
3.@Qualifier
4.@PropertySource
5.@Autowired
6.@Value
7.@Controller
8.@Service
9.@Repository
10.@Scope
11.@PostConstruct
12.@PreDestory
13.@Lazy
14.@Primary
15.@Import
16.@ImportResource
17.@Configuration
18.@Bean

Note: In layered apporach,having all the information of controller,service,dao in single xml file is not recomended,becoz it disturbs
the readablity if the projet is lengthy project.
To avoid this we need to seperate the xml configuration details w.r.t layers and all these layers are linked in single xml file.
To do this we use a tag called <import resource = ''/>

applicationContext.xml
======================
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:context="http://www.springframework.org/schema/context"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
        https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        https://www.springframework.org/schema/context/spring-context.xsd">

      <import resource="controller-beans.xml" />
      <import resource="service-beans.xml" />
      <import resource="persistence-beans.xml" />

</beans>
```

p-namespace and c-namespace
===========================
=> XMLSchema namespace is a library that contains set of xml tags.
=> every xml schema namespace is identified with uri/url

=> To use xml schema namespace in our xml file we need to import namespace uri/url in the xml file

namespace uri
context "http://www.springframework.org/schema/context"
beans
"http://www.springframework.org/schema/beans"
p
"http://www.springframework.org/schema/p"
c
"http://www.springframework.org/schema/c"

p namepsace is given as an alternative to <property name="" value='' ref=''/> to perform setter injection
c namespace is given as an alternative to <constructor-arg name='' value='' ref=''/> to perform constructor injection.

syntax:: <bean id='' class='' p:propertyname='' p:propertyname-ref=''/>
<bean id='' class='' c:propertyname='' c:propertyname-ref=''/>

applicationContext.xml
==========

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:p="http://www.springframework.org/schema/p"
      xmlns:c="http://www.springframework.org/schema/c"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
        https://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        https://www.springframework.org/schema/context/spring-context.xsd">

      <bean id="dob" class="java.util.Date">
            <property name="date" value="03" />
            <property name="month" value="0" />
            <property name="year" value="93" />
      </bean>

      <bean id="dos" class="java.util.Date" p:month="02" p:date="25"
            p:year="123" />

      <bean id="emp" class="in.ineuron.comp.Employee" p:eno="10"
            p:ename="sachin" p:dob-ref="dob" p:dept-ref="dept" />


      <bean id="dept" class="in.ineuron.comp.Department" c:dno="10"
            c:dname="CS" c:dos-ref="dos"/>

</beans>
```

Employee.java
==========

```
package in.ineuron.comp;

import java.util.Date;

public class Employee {
      private int eno;
```

```
	private String ename;
	private Date dob;
	private Department dept;

	setXXX(),toString()
}

Department.java
===============
package in.ineuron.comp;

import java.util.Date;

public class Department {

	private int dno;
	private String dname;
	private Date dos;

	public Department(int dno, String dname, Date dos) {
		System.out.println("Department:: Constructor Injection...");
		this.dno = dno;
		this.dname = dname;
		this.dos = dos;
	}

	@Override
	public String toString() {
		return "Department [dno=" + dno + ", dname=" + dname + ", dos=" + dos +
"]";
	}

}

TestApp.java
============
package in.ineuron.main;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import in.ineuron.comp.Employee;

public class TestApp {
	public static void main(String[] args) {

		ClassPathXmlApplicationContext factory = new
ClassPathXmlApplicationContext(
				"in/ineuron/cfg/applicationContext.xml");

		Employee employee = factory.getBean("emp", Employee.class);
		System.out.println(employee);

		factory.close();
	}
}
output
Department:: Constructor Injection...
Employee  :: Setter Injection...
Employee [eno=10, ename=sachin, dob=Sun Jan 03 10:28:30 IST 1993, dept=Department
```

[dno=10, dname=CS, dos=Sat Mar 25 10:28:30 IST 2023]]

Limitations of p-namespace and c-namespace
==========
1. It doesn't support collection collection injection
2. It doesn't resolve the problem of constructor injection using type,index,order
3. It came very lately when the industry was moving toward annotation driven programming.

DI Injection approach using XML + Annotation to perform CRUD operation on Database.
==========
=

IOCProj-27-DILayerApproachApp

Pure Java Configuaration(No XML)/ 100%Code driven SpringApp development/Java Config Approach of SpringApp development
==========
==========
ApplicationContext container
---

1. It is an extension of BeanFactory
2. Implementation classes of ApplicationContext(I)
    a. FileSystemXmlApplicationContext(standalone)
    b. ClassPathXmlApplicationContext(standalone)
    c. XmlWebApplicationContext(SpringMVC apps)
    d. AnnotationConfigApplicationContext(Standaloneapp's)
    e. AnnotationConfigWebApplicationContext(SpringMVC apps)

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*******");

	}
```

```
}
```