List of annotations used
==========
1.@Autowired
2.@Component
3.@Service
4.@Repository
5.@Qualifier
6.@Primary
7.@Lazy
8.@PropertySource
9.@Import
10.@ImportResource
12.@ComponentScan
13.@Configuration
14.@Value
15.@Bean
16.@PostConstruct
17.@PreDestroy

==========
1. @SpringBootApplication
|=>@Configuration,@ComponentScan,@EnableAutoConfiguration

Difference b/w Spring vs SpringBoot(interview question)
==========
1. Spring
It is a framework for JEE technologies/Application framework
The main feature is DependancyInjection and DependancyLookUp[JNDI].
It supports XML driven configuration as a inputs to the IOC-Container.
Programmer creates IOC container explicitly.
Allows to devleop spring apps using
a. XML
b. XML + Annotation
c. Pure Java(No XML)
Doesn't give embeded server to use in webapplications.
Doesn't give embeded database/inMemory Database
It is light weight because no autoconfiguration.
No support for "Microservices architecture" based application development.

2. SpringBoot
It provides abstraction for Spring framework and simplifies SpringApp development.
The main feature is AutoConfiguration(giving common things automatically)
Doesn't support XML driven configuration as a inputs to the IOC-Container.
Programmer doesn't create IOC container explicitly it gets created automatically using
SpringApplication.run().
Supports only one style of configuration that is AutoConfiguration where inputs are supplied
through application.properties/.yml file.
It gives embeded server(tomcat server,jetty server) to use in web applications.
It gives embeded database/InMemory database called "H2".
It is heavy weight because of AutoConfiguration.
Support of Microservices architecture is extensively avilable.

There are 2 different ways to perform injection to spring bean properties

a. @Value => It can be used to inject each value to spring bean properties
b. @ConfigurationProperties => It can be used to perform bulk injection.

eg:
application.properties
======================
org.info.companyName = ineuron
org.info.companyLoc = bengaluru
org.info.companyType = IT

using @Value
============
@Component("company)
@PropertySource("application.properties")
public class Company{

	@Value("${org.info.companyName}")
	private String name;

	@Value("${org.info.companyLoc}")
	private String adress;

	@Value("${org.info.copmanyType}")
	private String type;

	toString()
}

using @ConfigurationProperties
==============================
@Component("company)
@ConfigurationProperites(prefix= "org.info")
public class Company{
	private String companyName;
	private String companyLoc;
	private String companyType;

	setXXXX(),toString()
}

What is the difference b/w @Value and @CofigurationProperties?
@Value
	=> It is given by Spring framework,so it can be used in Spring and SpringBoot applications.
	=> Support single value injection to Spring bean property.
	=> It performs field level injection(setters not required)
	=> Common prefix of all keys are not required in application.properties/application.yml file
	=> Keys in properties file and property names need not match.
	=> If specified key is not present then it would result in "IllegalArgumentException".

		@Value("${user.info.age}");
	syntax:: public int age;

@ConfigurationProperties
	=> It is given by SpringBoot framework,so it can be used only SpringBoot applications.
	=> Support bulk operation

```
      => It perform setter level injection internally, so setters are mandatory
      => Common prefix of all keys are required in
application.properties/application.yml file.
      => keys in properties file and property names should match
      => If the matching key is not found then it would neglect the injection.


Note: While working with @ConfigurationProperties, it is always suggested to add
configuarationProcessor inside pom.xml file
<dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-configuration-processor</artifactId>
      <optional>true</optional>
</dependency>

Behind the scenes
=================
DAO
===
  @Autowired
  private DataSource dataSource;  //spring-boot-starter-jdbc[hikaricp]

application.properties
======================
   spring.datasource.url = jdbc:mysql:///octbatch
   spring.datasource.username=root
   spring.datasource.password= root123


@ConfigurationProperites(prefix = "spring.datasource")
class .....{
      private String url;
      private String username;
      private String password;

      setXXX(),toString()
}

Note:
If we try to inject two different values to same spring bean property using both
@Value and @Configurationproperties, which value will
be injected?
@Value ===> Field injection [ object created, and using reflection api our value
will be set]
@ConfigurationProperties===> Setter Injection[ojbect created,@Value-> field
injection,@ConfigurationProperties->Setterinjection]
      Since @ConfigurationProperties performs setter injection, so it overrides the
field injection value given by @Value annotation.


Usage of SPEL in @Value Annotation
==================================

application.properties
----------------------
item.dosa.price = 40
item.idli.price = 20
item.vada.price = 10
```

```
@Component(value = "info")
public class ItemsInfo {

	@Value("${item.idli.price}")
	public float idlyPrice;

	@Value("${item.vada.price}")
	public float vadaPrice;

	@Value("${item.dosa.price}")
	public float dosaPrice;
}

BillGenerator.class
==================
@Component("bill")
public class BillGenerator {

	@Value("#{info.idlyPrice+info.vadaPrice+info.dosaPrice}")
	private Float billAmount;//SPEL[SpringExpressionLanguage] is used

	@Value("A2B")
	private String hotelName;

	@Autowired
	private ItemsInfo info;
}

Application.java
----------------

package in.ineuron;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ConfigurableApplicationContext;
import in.ineuron.comp.BillGenerator;

@SpringBootApplication
public class Application {

	public static void main(String[] args) {
		ApplicationContext context = SpringApplication.run(Application.class,
args);

		BillGenerator billGenerator = context.getBean(BillGenerator.class);
		System.out.println(billGenerator);

		((ConfigurableApplicationContext) context).close();
	}
}

Output
BillGenerator [billAmount=70.0, hotelName=A2B, info=ItemsInfo [idlyPrice=20.0,
vadaPrice=10.0, dosaPrice=40.0]]


Injection of all types from properties file
==========================================
```

1. Primtive type
2. Array type
3. List type
4. Set type
5. Map type
6. HAS-A type

Injecting values to different types like Arrays,List,Set,Map,HAS-A Property of SpringBean using Properties/.yml file
==========
==========
=> The allowed special characters in properties file is ".","-","[]".
=> To work with Array,List,Set we need to use prefix.<property-name>[index]=value //index should be sequential
=> To work with Map<K,V> we need to use prefix.<property-name>.<key>=<value>.

Company.java
==========
```
@Component(value = "company")
public class Company {
	private String type;
	private String location;
	private String name;

}
```

Employee.java
==========
```
@Component("employee")
@ConfigurationProperties(prefix="emp.info")
public class Employee {
	private String name;
	private Integer id;
	private String[] nickNames;
	private List<String> teamMembers;
	private Set<Long> phoneNumbers;
	private Map<String, Object> idDetails;

	@Autowired
	private Company company;
}
```

application.properties
==========
```
emp.info.name=sachin
emp.info.id=10

#Array properties
emp.info.skills[0] = java
emp.info.skills[1] = jee
emp.info.skills[2] = ORM
emp.info.skills[3] = SpringBoot

#List Properties
emp.info.team-members[0] = sauav
emp.info.team-members[1] = dravid
emp.info.team-members[2] = yuvraj
```

```
#Set Properties
emp.info.phone-numbers[0]=9998887776
emp.info.phone-numbers[1]=6667778885
emp.info.phone-numbers[2]=4445556667

#Map properties
emp.info.id-details.adharNo= 55566765
emp.info.id-details.panNo= 2321234
emp.info.id-details.voterid= 44323456

#HAS-A properties
emp.info.company.title= PW Skills
emp.info.company.location= Delhi
emp.info.company.size= 5000
```

```
Application.java
================
package in.ineuron;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ConfigurableApplicationContext;
import in.ineuron.comp.Employee;

@SpringBootApplication
public class Application {

      public static void main(String[] args) {
            ApplicationContext context = SpringApplication.run(Application.class,
args);

            Employee employee = context.getBean(Employee.class);
            System.out.println(employee);

            ((ConfigurableApplicationContext) context).close();
      }
}

output
Employee [name=sachin, id=10,
             skills=[java, jee, ORM, SpringBoot],
             teamMembers=[sauav, dravid, yuvraj],
              phoneNumbers=[9998887776, 6667778885, 4445556667],
              idDetails={adharNo=55566765, panNo=2321234, voterid=44323456},
              company=Company [title=PW Skills, location=Delhi, size=5000]
            ]


YML/YAML Injection
==================
=> It stands for Yet Another MarkUp Language
=> The extension of the file is .yml or .yaml
=> The biggest limitation of properties file is nodes/level will be repeated in
mulitple keys,especially while working with
   common prefix concepts like collection,HAS-A property to support bulk injection
using @ConfigurationProperties.
```

=> SpringFramework doesnot support yml file/where as SpringBoot support yml injection
=> SpringBoot framework internally use snakeyml<ver>.jar for processing the yml file.

application.properties
======================
```
emp.info.id=10
emp.info.name=sachin
emp.info.loc=MI
```

application.yml
==============
```
emp:
   info:
      id: 10
      name: sachin
      loc: MI
```

Rules while writing yml file
============================
 => same nodes/level in the key should not be duplicated.
 => replace "." of each node/level with ":" and write new node in next line with proper indentation(minimum single space is required)
 => replace "=" symbol with ":" before placing value having minimum single space.
 => To replace Array,List,Set elements use "-".
 => Take Map collection keys and HAS-A property subkeys as the new nodes/levels.
 => use #symbol for Commenting.

application.yml
===============
```
#Array properties
emp:
   info:
      skills:
      - java
      - jee
      - ORM
      - SpringBoot

#List Properties
       team-members:
       - sauav
     - dravid
     - yuvraj

#Set Properties
       phone-numbers:
       - 9998887776
       - 6667778885
     - 4445556667

#Map properties
     id-details:
        adharNo: 55566765
        panNo: 2321234
        voterid: 44323456
```

```
#HAS-A properties
     company:
        title: PW Skills
        location: Delhi
        size: 5000

eg#2.
application.properties
======================
spring.datasource.url=jdbc:mysql:///octbatch
spring.datasource.username=root
spring.datasource.password=root123

application.yml
===============
spring:
   datasource:
      url: jdbc:mysql:///octbatch
       username: root
       password: root123

=> Once we have properties file in eclipse, we can convert into yml using sts
supplied plugin.
=> The nodes/level in the keys of properties file/.yml file are not case sensitive.

What is the difference b/w properties file and .yml file?

Properties file
===============
=> no rules and guideliness to develop properties file,just Key=Value
=> it can be used only in java
=> No way related to json format
=> can be used in both Spring and SpringBoot project
=> nodes/level in the keys can be duplicated.
=> it is not a hierarchial data
=> Custom properties file can be injected to bean using @PropertySource
=> While working with profiles in springboot we can't place multiple profiles in
single properties file.
=> Spring/SpringBoot directly loads and reads the content of properties file.
=> use properties file when no of keys are minimal and nodes/level in the key are
not duplicated.

YML file
========
=> specification/rule and guideliness given by www.yml.org
=> can be used in .java,.ruby,.python etc
=> Super set of JSON
=> Supported only by SpringBoot
=> nodes/level in the keys can't be duplicated.
=> Its a hierarchial data
=> Custom files will be configured using @PropertySource and specifying
PropertySource class is required.
=> we can place multiple profiles in single yml file having seperation with "--".
=> every yml file will be converted to property files before loading.
=> use yml file when no of keys are more and nodes/level in the key are repeating.
```