Go through SpringCore basic questions(5 minutes)

Topics
 a. Spring DI
      1. Setter injection
      2. Constructor injection
      3. Field injection
      4. LMI(Look up Method Injection)
 b. Importance of Scope for a bean
      a. prototype
      b. singleton
      c. request
      d. session
 c. What is Autowiring and explain the type of Autowiring?
 d. How to resolve the problems like
      a. NoSuchBeanDefnitionException
      b. NoUniqueBeanDefnitionException
 e. Explain how IOC container works?
 f. Explain all the annotation used in SpringCore
      a. @Component
      b. @Configuration
      c. @Bean
      d. @ComponentScan
        e. @Value
      f. @Autowired
      g. @Service
      h. @PropertySource
      i. @Lazy
      j. @Qualifier
      k. @Scope
      l. @Primary

  g. Annotations w.r.t SpringMVC
      a. @Controller
      b. @Service
      c. @Repository
      d. @GetMapping
      e. @PathVariable
      f. @PostMapping
      g. @ModelAttribute
      h. @RequestParam

  h. Control flow in SpringMVC applications
      a. DispatcherServlet
      b. HandlerMapping
      c. Controller
      d. ViewResolver

  i. SpringDataJPA supplied annotations
      a. @Repository
            1.JpaRepository
            2.CrudRepository
            3.PagingAndSortingRepository
         Note: Usage of Optional<T> api is mandatory
            4. Writing customized query using findByXXXXX() approach.
      b. @Transactional
        c. @Query

++++++++++

SpringCore
++++++++++
  Why should we use Spring framework to build applications?
      Framework => LOC[Lines of Code] is less and promotes faster development.

Benefits of working with SpringFramework
      a. Dependancy injection
            => The process of injecting dependent object into target object and
making the target object ready for buisness use
               is refered as "Dependancy injection".
        How many ways of dependancy injection is possible from SpringFramework?
            a. Setter injection.
            b. Constructor injection.
            c. Field injection.

+++++++++++++++++
Setter injection
+++++++++++++++++
 => While creating a target object, dependant object will be injected by the
container in setter style, this type of DI is called as
    "Setter Injection".

Note: If we write a code using jdk api's in real time
            a. we need to manage the objects creation till its destruction.
            b. we need to make a call to methods explicitly.
            c. control flow of the application is decided by the developer.

For an Enterprise application managing the above 3 steps is so costly to avoid this
, we take the help of "IOC".

++
Setter injection for Collection type of objects
++
pom.xml
=======
```
<properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
</properties>

<dependencies>
      <!-- https://mvnrepository.com/artifact/org.springframework/spring-context --
>
      <dependency>
          <groupId>org.springframework</groupId>
          <artifactId>spring-context</artifactId>
          <version>5.3.29</version>
      </dependency>

</dependencies>
```

+++++++++++++++
AppConfig.java
+++++++++++++++
```
package in.ineuron.config;

import java.util.ArrayList;
import java.util.HashMap;
```

```java
import java.util.LinkedHashSet;
import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.Set;

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

import in.ineuron.bean.Product;

@Configuration
public class AppConfig {

	// 1 method = 1 bean
	@Bean
	public Product prodObj() {

		Product product = new Product();
		product.setData(list());
		product.setModels(set());
		product.setModes(map());
		product.setContext(props());
		return product;
	}

	public List<String> list() {
		ArrayList<String> list = new ArrayList<String>();
		list.add("p1");
		list.add("p2");
		list.add("p3");
		list.add("p4");
		return list;
	}

	public Set<String> set() {
		LinkedHashSet<String> set = new LinkedHashSet<String>();
		set.add("S1");
		set.add("S2");
		set.add("S3");
		set.add("S4");
		return set;
	}

	public Map<Integer, String> map() {
		HashMap<Integer, String> map = new HashMap<Integer, String>();
		map.put(10, "A");
		map.put(20, "B");
		map.put(30, "C");
		return map;
	}

	public Properties props() {
		Properties props = new Properties();
		props.put("A1", "B1");
		props.put("A2", "B2");
		props.put("A3", "B3");
		return props;
	}
```

```
}

++++++++++
Product.java
++++++++++++
package in.ineuron.bean;

import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.Set;

public class Product {

	private List<String> data;
	private Set<String> models;
	private Map<Integer, String> modes;
	private Properties context;

	public List<String> getData() {
		return data;
	}

	public void setData(List<String> data) {
		this.data = data;
	}

	public Set<String> getModels() {
		return models;
	}

	public void setModels(Set<String> models) {
		this.models = models;
	}

	public Map<Integer, String> getModes() {
		return modes;
	}

	public void setModes(Map<Integer, String> modes) {
		this.modes = modes;
	}

	public Properties getContext() {
		return context;
	}

	public void setContext(Properties context) {
		this.context = context;
	}

	@Override
	public String toString() {
		return "Product [data=" + data + ", models=" + models + ", modes=" +
modes + ", context=" + context + "]";
	}

}
```

```
++++++++++++
TestApp.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.bean.Product;
import in.ineuron.config.AppConfig;

public class TestApp {

	public static void main(String[] args) {

		System.out.println("STARTING THE CONTAINER");
		ApplicationContext context = new
AnnotationConfigApplicationContext(AppConfig.class);
		System.out.println("CONTAINER STARTED");
		Product product = context.getBean("prodObj", Product.class);
		System.out.println(product);
		((AbstractApplicationContext) context).close();
		System.out.println("STOPPING THE CONTAINER");
	}
}

Output
STARTING THE CONTAINER
CONTAINER STARTED
Product [data=[p1, p2, p3, p4], models=[S1, S2, S3, S4], modes={20=B, 10=A, 30=C},
context={A1=B1, A2=B2, A3=B3}]
STOPPING THE CONTAINER

++++++++++++++++++++++
Pure Java Configuration
++++++++++++++++++++++
package in.ineuron.bean;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

@Component("student")
public class Student {

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

	public Student() {
		System.out.println("Student object is instantiated...");
	}

	@Value("10")
	private Integer sid;

	@Value("sachin")
	private String sname;
```

```
	@Value("MI")
	private String saddress;

	@Override
	public String toString() {
		return "Student [sid=" + sid + ", sname=" + sname + ", saddress=" +
saddress + "]";
	}

}
```

++++++++++++++
AppConfig.java
++++++++++++++

```
package in.ineuron.config;

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

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

}
```

++++++++++++
TestApp.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.bean.Student;
import in.ineuron.config.AppConfig;

public class TestApp {

	public static void main(String[] args) {
		System.out.println("CONTAINER STARTED....");
		ApplicationContext context = new
AnnotationConfigApplicationContext(AppConfig.class);

		System.out.println("ASKING CONTAINER TO GIVE STUDENT BEAN");
		Student student = context.getBean("student", Student.class);
		System.out.println(student);
		System.out.println("CLOSING  THE CONTAINER...");
		((AbstractApplicationContext) context).close();

	}

}
```

output

```
CONTAINER STARTED....
Student.class file is loading...
Student object is instantiated...
```

```
ASKING CONTAINER TO GIVE STUDENT BEAN
Student [sid=10, sname=sachin, saddress=MI]
CLOSING  THE CONTAINER...
```

@Component => It is used to create an object and store in IOC container.
@ComponentScan => It is used to inform the IOC container to scan for the components in the following packages.
@Configuration => It is used to inform the IOC container that the corresponding class provides the configuration details.

What is the difference b/w @Component vs @Bean?
  @Component -> It is used to create an object,but this annotation is applied only for userdefined class
            It is applied only at class level.

  @Bean   -> It is used to create an object,but this annotation is applied only for predefined class.
            It is applied only at method level.

What is the difference b/w @Component vs @ComponentScan?
      Can't be compared b/w @Component => for object creation,@ComponentScan-> scan for the classes to create an object.

 @Value => It is meant for injecting the values to the variables(statically injected)

++
+
Getting the values dynamically from application.properties file(src/main/resources)
++
+
1. Product.java

```
package in.ineuron.bean;

public class Product {

      private Integer pid;
      private String pname;
      private String ptype;

      public Integer getPid() {
            return pid;
      }

      public void setPid(Integer pid) {
            this.pid = pid;
      }

      public String getPname() {
            return pname;
      }

      public void setPname(String pname) {
            this.pname = pname;
      }

      public String getPtype() {
```

```
		return ptype;
	}

	public void setPtype(String ptype) {
		this.ptype = ptype;
	}

	@Override
	public String toString() {
		return "Product [pid=" + pid + ", pname=" + pname + ", ptype=" + ptype
+ "]";
	}

}
```

2. AppConfig.java

```
package in.ineuron.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.core.env.Environment;

import in.ineuron.bean.Product;

@Configuration
@PropertySource("classpath:application.properties")
public class AppConfig {

	@Autowired
	private Environment environment;


	// 1 bean = 1 method
	@Bean
	public Product pobj() {
		Product product = new Product();
		product.setPid(environment.getProperty("pid", Integer.class, null));
		product.setPname(environment.getProperty("pname"));
		product.setPtype(environment.getProperty("ptype"));
		return product;
	}

}
```

3. application.properties

```
pid = 10
pname = fossil
ptype = chronography
```

4. TestApp.java

```
package in.ineuron.test;

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

import in.ineuron.bean.Product;
```

```
import in.ineuron.config.AppConfig;

public class TestApp {

	public static void main(String[] args) {

		ApplicationContext context = new
AnnotationConfigApplicationContext(AppConfig.class);

		Product product = context.getBean("pobj", Product.class);
		System.out.println(product);

		((AbstractApplicationContext) context).close();

	}

}

output
 Product [pid=10, pname=fossil, ptype=chronography]

++++++++++++++++++++
usage of @Autowired
++++++++++++++++++++
1. Product.java
package in.ineuron.bean;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

@Component
public class Product {

	@Value("10")
	private Integer pid;

	@Value("25000")
	private Integer pcost;

	@Autowired
	private Model model;

	public Integer getPid() {
		return pid;
	}

	public void setPid(Integer pid) {
		this.pid = pid;
	}

	public Integer getPcost() {
		return pcost;
	}

	public void setPcost(Integer pcost) {
		this.pcost = pcost;
	}
```

```
	public Model getModel() {
		return model;
	}

	@Override
	public String toString() {
		return "Product [pid=" + pid + ", pcost=" + pcost + ", model=" + model
+ "]";
	}

}
```

2. Model.java

```
package in.ineuron.bean;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

@Component
public class Model {

	@Value("100")
	private Integer mid;

	@Value("chronography")
	private String type;

	public Integer getMid() {
		return mid;
	}

	public void setMid(Integer mid) {
		this.mid = mid;
	}

	public String getType() {
		return type;
	}

	public void setType(String type) {
		this.type = type;
	}

	@Override
	public String toString() {
		return "Model [mid=" + mid + ", type=" + type + "]";
	}

}
```

3. AppConfig.java

```
package in.ineuron.config;

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

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

```
}

4. 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.bean.Product;
import in.ineuron.config.AppConfig;

public class TestApp {

	public static void main(String[] args) {

		ApplicationContext context= new
AnnotationConfigApplicationContext(AppConfig.class);
		Product product = context.getBean("product",Product.class);
		System.out.println(product);
		((AbstractApplicationContext) context).close();
	}
}
Output
Product [pid=10, pcost=25000, model=Model [mid=100, type=chronography]]
```