1. ***Configuring scope to Class***

@Scope(ConfigurableBeanFactory.SCOPE\_PROTOTYPE)

1. ***Configuring Proxy for scopes (one is singleton, other as prototype)***

@Scope(value=ConfigurableBeanFactory.SCOPE\_PROTOTYPE, proxyMode =ScopedProxyMode.TARGET\_CLASS)

1. ***Making a Bean Qualifying for more than one implementations***
2. @Primary
3. @Qualifier
4. Reference variable for an Interface as ClassName i.e, Using Dependency By Name

Singleton

* Gang Of Four : Single Instance per JVM
* Spring : Single Instance per Application Context

ComponenScan will be automatically checked in default package where Spring Boot Application is present

***Converting from Spring Boot to Spring***

<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-core</artifactId>  
</dependency>

<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
</dependency>

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
</dependency>

@Configuration

@ComponentScan(basePackages="com.mytrah")

@SpringBootApplication

Remove

SpringApplication.*run*

ApplicationContext ac = new AnnotationConfigApplicationContext(ExcelApplication.class);

Add Slf4J Dependency

<dependency>  
 <groupId>org.slf4j</groupId>  
 <artifactId>slf4j-api</artifactId>  
</dependency>

<dependency>  
 <groupId>ch.qos.logback</groupId>  
 <artifactId>logback-classic</artifactId>  
</dependency>

We need to close Application Context i.e, AnnotationConfigApplicationContext.close() or We can use it in try block which automatically closes

(Container and Dependency Inject) CDI is Java EE Standard –

1. @Named (@Component)
2. @Inject (@Autowired)
3. @Qualifier (@Primary)

**Bean Factory** provides Basic management for beans and wiring of dependencies. It can be used where memory is a constraint(ex. IOT)

**Application Context** provides Spring AOP features , I18N, WebApplicationContext for Web Applications. In all typical Enterprise applications.

**Stereo Types**

@Component – Generic Component. It can be used in any layers (Web, Business, Data Layers)

@Controller – Defines a Controller in Web layer (MVC Pattern)

@Service – Business Service Façade

@Repository – Encapsulating storage, retrieval, and search behavior typically from relational Database. (Eg. Spring provides default Exception Translation facility If you use @Repository)

Lot of JDBC Exceptions. Spring classifies them and translates them and this feature provided if you are using @Repository

By seeing @Service, we can use AOP Annotation and we can log all content which is coming in.

@Value – Used to read a value from an properties file

Eg. @Value(“${external.service.url}”)

@PropertySource(“classpath:app.properties”) – Annotation used to load a Customized property file. Use it on top of the Class Name.

Junit:

Absence of failure is success

*assertEquals*(1, 1);

boolean condition =true;

*assertTrue*(condition);

assertFalse(condition);

assertNotNull(ob);\*/

*assertArrayEquals*(arr1, arr2);

@Before – Annotation Applied to a method, which will be executed before executing any test method

@After – Annotation Applied to a method, which will be executed after executing any test method. Eg. Performing some cleaning activities.

@BeforeClass – Annotation Applied to a static method, which will be executed before executing any test Class. Eg. Establishing a Database Connection

@After – Annotation Applied to a static method, which will be executed after executing any test Class.

Mockito :

When we are using Mocks we don’t really need to create multiple versions of Classes. Mocks make it very easy to dynamically create different classes and make them to return the data that we would want to return.

The following static methods are used

**mock**(InterfaceType.class)

**when**(interfaceTypeReference.method).thenReturn(provide Data as required);

**thenReturn** :

DataService dataServiceMock = *mock*(DataService.class); *when*(dataServiceMock.retrieveData()).thenReturn(new int[] {});

@Mock : Used on a Interface Type for which Mocking can be done.

Eg. *@Mock* DataService dataServiceMock;

@InjectMocks : Implemetion will be created for Interface and injected directly

Eg. *@*InjectMocksDataServiceImpl dataServiceImpl;

@RunWith(MockitoJUnitRunner.class) : Used on Classs Type

*@RunWith*(MockitoJUnitRunner.class) public class GreatestTestUsingMockitoAnnotations {

For Specific Parameter:

*when*(listMock.get(0)).thenReturn("Some String");

**Mockito.anyInt**() : For Generic Parameter (if List Class is trying to get any Index then return Something)

Eg. *when*(listMock.get(Mockito.*anyInt*())).thenReturn("Some String");

We can not use one Generic Parameter and one specific parameter.

Eg.

*when*(listMock.get(Mockito.*anyInt*())).thenReturn("Some String");

*when*(listMock.get(0)).thenReturn("Zero Value");

*assertEquals*("Zero Value", listMock.get(0));

*assertEquals*("Some String", listMock.get(1));

To load the Context we use **@RunWith(SpringRunner.class), @ContextConfuguration(class=XYZ.class)**

**Context Configuration :** It is part of Spring Test Module. It helps us to run a configuration. To load the context, we use this and specify the **Spring Boot Application** Class (Spring Application which is used to start the Application and contains @Configuration, @Component Annotations).

Eg. @ContextConfiguration(classes=MytrahExcelApplication.class)

Eg. @ContextConfiguration(locations=”/applicationContext.xml”) (It would load all the beans specified in applicationContext.xml)

**@RunWith(SpringRunner.class)** : It would launch the configuration up.

**SpringRunner.class** : It is a part of Spring Test Module.

To allow CORS, we use the following on the top of the controller

@CrossOrigin(origins = "\*", allowedHeaders = "\*")

To allow CORS globally, see: https://spring.io/guides/gs/rest-service-cors/

@Bean

public WebMvcConfigurer corsConfigurer() {

return new WebMvcConfigurerAdapter() {

@Override

public void addCorsMappings(CorsRegistry registry) {

registry.addMapping("/greeting-javaconfig").allowedOrigins("http://localhost:9000");

}

};

}