Q.what is spring ?
Ans.Spring is java based framework used to develop enterprize application,web-application,cloud-based applications, mobile applications,etc.
Spring framework helps in developing a loosely coupled application that is simple, easily testable, reusable, and maintainable.

Q.what is spring boot ?
Ans.It is easy way to configure spring application (initially it was tedious job to configure) and developer can docus on business logic

Q.Whats new in Spring 5?
Ans.Relese of Spring 5--->sep 2017
Whats new in Spring 5:
1.updated min requirement with java 8 or above
2.Added new reactive programming framework :Spring webflux
3.update spring mvc to use new version of servlet Api 4.0
4.Functional programming using Kotlin language support
5.Testing improvements by supporting integration with JUnit5

Q.structure of enterprise application ?
Ans.DB-->PERSISTENCE LAYER-->SERVICE LAYER--->PRESENTATION LAYER

Q.Why Spring ?
Ans.1.Object Creation and maintaining its life cycle
2.Injecting Dependencies
3.helps to develop of losely coupled application
4.Easily testable by mocking objects

```
public class CustomerServiceImpl implements CustomerService{
CustomerRepository customerRepository= new
CustomerRepositoryImpl();****FOCUS******
    public String createCustomer(CustomerDto dto) {
            return customerRepository.createCustomer(dto);

      }
      public String fetchCustomer() {
            return customerRepository.fetchCustomer();
      }
}
```

CustomeServiceImpl depends on CustomerRepository. It also instantiates CustomerRepositoryImpl class which makes it tightly coupled with CustomerRepositoryImpl class. This is a bad design because of the following reasons:
1.If you want to unit test for createCustomer() method then you need a mock object of CustomerRepository. But you cannot use a mock object because there is no way to substitute the CustomerRepositoryImpl object that CustomerServiceImpl has. So testing CustomerServiceImpl becomes difficult.
2.Also, you cannot use a different implementation of CustomerRepository other than CustomerRepositoryImpl because of tight coupling.

Uff !!! Need more flexible solution .Check below code

```
public class CustomerServiceImpl implements CustomerService {
    private CustomerRepository customerRepository;****FOCUS******
```

```
    public CustomerServiceImpl(CustomerRepository customerRepository)
{****FOCUS******
        this.customerRepository = customerRepository;
    }
     public String createCustomer(CustomerDto dto) {
            return customerRepository.createCustomer(dto);

      }
 }
```

The above code resolves both problems that are mentioned above .But this would be tedious to manually wire dependency and change vast amount of code .
So what is the solution?
Let external framework will take care of this.so creation of objects and injecting dependency will be done by external framework called Dependency Injection Framework(also called IoC containers).

Q.What is Inversion of Control(IoC)?
Ans.creation of objects and injecting dependency will be done by external framework .This process is called IoC(reversal of responsibility)

Q.Benefits of Dependency Injection(DI):
Ans.-Helps to create loosely coupled application which makes it re-usability and easy testing
-Allows to replace actual objects with mock objects for testing, this improves testability

Q.What are features of spring framework?[LLISA]
Ans.Light Weight -->Spring JARs are relatively small + It can be deployed in Tomcat and they do not require heavy-weight application servers.
Loosely Coupled
Inversion of Control(IoC)
Spring Container-->Spring Container takes care of object creation, initialization, and managing object dependencies
Aspect Oriented Programming(AOP)-->logging,transaction,security is separated from core business logic

Q.Spring Modules ?
Ans.Spring Framework 5.0 has the following key module groups:
Core Container: These are core modules that provide "key features" of the Spring framework.
Data Access/Integration: These modules support JDBC and ORM data access approaches in Spring applications.
Web: These modules provide support to create web applications.
Others: Spring also provides few other modules such as the Test for testing Spring applications.

1.Core Container:
Core:This is the key module of Spring Framework which provides fundamental support on which all other modules of the framework are dependent.
Bean: This module provides a "basic Spring container" called BeanFactory.
Context: This module provides one more Spring container called ApplicationContext which inherits the basic features of the BeanFactory container and also provides additional features to support "enterprise application development".

AOP (Aspect Oriented Programming) and aspects: These modules help in isolating "cross-cutting(logging, security,transactions) functionality" from business logic.

2.Data Access/Integration:
Java Database Connectivity (JDBC): It provides an abstract layer to support JDBC calls to relational databases.
Object Relational Mapping (ORM): It provides integration support for popular ORM(Object-Relational Mapping) solutions such as Hibernate, JPA, etc.

3.Web:
Web: This module has a container called "web application context" which inherits basic features from ApplicationContext container and adds features to develop web based applications.
Webmvc: It provides the implementation of the MVC(model-view-controller).also supports features to implement RESTful Web Services.
WebFlux: Spring 5 introduces Spring WebFlux to support Reactive programming.

4.Others:
Test: This module provides the required support to test Spring applications using TestNG or JUnit.

Q.What are beans or spring beans ?
Ans.The classes whose objects life cycle is managed by Spring(container manages) are called as beans or Spring beans

Q.What is Spring IoC container?
Ans.Spring IoC container used to create and inject required objects(DI).
The Spring IoC container "is represented" by following interfaces:
1.The BeanFactory interface represents the basic container which provides basic functionalities.
It "instantiates bean" whenever asked for by the client application.
Using its getBean() method you can get instances of beans.
2.The ApplicationContext interface extends BeanFactory interface
3.There are many implementations of this interface.Commonly used are:
ClassPathXmlApplicationContext : It is used for Java applications using XML based configuration.
AnnotationConfigApplicationContext : It is used for Java applications using annotations based configuration.

POJO classes + configuration metadata-----> IoC container---creates------->objects required by the application ***FOCUS******

In a Spring Application, configuration metadata can be provided in following ways
XML configuration
Annotation-based configuration
Java-based configuration

Q.Difference between BeanFactory and ApplicationContext ?
Ans. BeanFactory

ApplicationContext

1.It does not support "annotation based DEPENDENCY INJECTION".
Support annotation based Dependency Injection.
2.It does not support enterprise SERVICES.
Support enterprise services such as validations, internationalization, etc.
3.By default, it supports Lazy Loading.
By default, it supports Eager Loading. Beans are instantiated during load time.

4.//Loading BeanFactory
// Loading ApplicationContext and instantiating bean
BeanFactory factory = new AnnotationConfigApplicationContext(SpringConfiguration.class);
ApplicationContext context = new AnnotationConfigApplicationContext(SpringConfiguration.class);
// Instantiating bean during first access using getBean()
// Instantiating bean during first access using getBean()
CustomerServiceImpl service = (CustomerServiceImpl) factory.getBean("customerService"); CustomerServiceImpl service = (CustomerServiceImpl) context.getBean("customerService");

Q.What are different ways of accessing Beans?
Ans.1.CustomerServiceImpl service = (CustomerServiceImpl) context.getBean("customerService");
2.CustomerServiceImpl service = context.getBean(CustomerServiceImpl.class);
3.CustomerServiceImpl service = context.getBean("customerService", CustomerServiceImpl.class);

Q.Define @Configuration ?
Ans.Class marked with this annotation is considered as configuration class, and it's expected to contain details on beans that are to be created in the Spring application context.

Q.Define @Bean?
Ans.1.It is defined on methods
2. it is used to create instance of bean
Example:

```
@Configuration
public class SpringConfiguration {

	@Bean
	public CustomerServiceImpl customerService() {

		return new CustomerServiceImpl();
	}
}
```

Q.Naming of bean when @Bean is used ?
Ans.By default, the bean name is the same as the name of the method.So in the above code bean name is customerService.If you want to change the bean name then we can use "name" attribute. ****FOCUS******
Example:

```
@Configuration
public class SpringConfiguration {
```

```
	@Bean(name="service")
	public CustomerServiceImpl customerService() {

		return new CustomerServiceImpl();
	}
}
```

Q.What are ways to initialize the properties ?

Ans.Spring container uses one of these two ways to initialize the properties:

Constructor Injection: This is achieved when the container invokes a "parameterized constructor" to "initialize the properties of a class"

Setter Injection: This is achieved when the container invokes setter methods of a class to initialize the properties "after invoking a default constructor".

1.Example for CONSTRUCTOR INJECTION:

```
//Customer Rrepository class
public class CustomerRepository {

	public String fetchCustomer(int count) {
		return " The no of customers fetched are : " + count;
	}
	public String createCustomer() {
		return "Customer is successfully created";
	}
}

//Customer service interface
public interface CustomerService {
	public String fetchCustomer();
	public String createCustomer();
}

//CustomerServiceImpl class
public class CustomerServiceImpl implements CustomerService {
	private int count;
	private CustomerRepository repository;
	public CustomerServiceImpl(CustomerRepository repository, int
count) {
		this.count = count;
		this.repository = repository;
	}
	public String fetchCustomer() {
		return repository.fetchCustomer(count);
	}
	public String createCustomer() {
		return repository.createCustomer();
	}
}

//configuration class
@Configuration
public class SpringConfiguration {
	@Bean
	public CustomerServiceImpl customerService() {
		return new CustomerServiceImpl(customerRepository(), 20);
	}
```

```
	@Bean
	public CustomerRepository customerRepository() {
		return new CustomerRepository();
	}
}

//
public class Client {
	public static void main(String[] args) {
		CustomerServiceImpl service = null;
		ApplicationContext context = new
AnnotationConfigApplicationContext(SpringConfiguration.class);
		service = (CustomerServiceImpl)
context.getBean("customerService");
		System.out.println(service.fetchCustomer());
		context.close();
	}
}

2.Example for SETTER INJECTION:
//Customer Rrepository class
public class CustomerRepository {
	public String fetchCustomer(int count) {
		return " The no of customers fetched are : " + count;
	}
	public String createCustomer() {
		return "Customer is successfully created";
	}
}

////Customer service interface
public interface CustomerService {
	public String fetchCustomer();
	public String createCustomer();
}

//CustomerServiceImpl class
public class CustomerServiceImpl implements CustomerService {
	private int count;
	private CustomerRepository repository;
	public CustomerServiceImpl() {
	}
	public void setCount(int count) {
		this.count = count;
	}
	public void setRepository(CustomerRepository repository) {
		this.repository = repository;
	}
	public String fetchCustomer() {
		return repository.fetchCustomer(count);
	}
	public String createCustomer() {
		return repository.createCustomer();
	}
}

//configuration class
@Configuration
```

```
public class SpringConfiguration {
	@Bean // Setter Injection
	public CustomerRepository customerRepository() {
		CustomerRepository customerRepository = new
CustomerRepository();
		return customerRepository;
	}
	@Bean // Setter Injection
	public CustomerServiceImpl customerService() {
		CustomerServiceImpl customerService = new
CustomerServiceImpl();
		customerService.setCount(10);
		customerService.setRepository(customerRepository());
		return customerService;
	}
}

public class Client {
	public static void main(String[] args) {
		CustomerServiceImpl service = null;
		AbstractApplicationContext context = new
AnnotationConfigApplicationContext(SpringConfiguration.class);
		service = (CustomerServiceImpl)
context.getBean("customerService");
		System.out.println(service.fetchCustomer());
		context.close();
	}
}
```

Q.Why AutoScanning is Required ?
Ans.It scans packages and create objects/beans

```
	@Configuration
	@ComponentScan(basePackages="com.infy")
	public class SpringConfiguration {
	}
```

Note:we have to define @ComponentScan to enable component scan .Spring do not scan automatically.
Note:by default ,Spring will scan the package that contains Configuration class and it subpackages for beans.
But if you want to scan a different package or multiple packages then you can specify this with the basePackages attribute as follows:

```
@Configuration
@ComponentScan(basePackages = "com.infy.service,com.infy.repository")
public class SpringConfiguration {
}
```

Q.what are Stereotype annotations?
Ans.Stereotype annotations are @Component, @Service, @Repository, and @Controller annotations.
These annotations are used for auto-detection of beans using @ComponentScan.
The Spring stereotype @Component is the parent stereotype.
he other stereotypes are the specialization of @Component annotation

@Component---->It indicates the class(POJO class) as a Spring component.
@Service---->It indicates the Service class(POJO class) in the business layer.

@Repository---->It indicates the Repository class(POJO class in Spring DATA) in the persistence layer.
@Controller---->It indicates the Controller class(POJO class in Spring MVC) in the presentation layer.
Note:@Component should be used when your class does not fall into either of three categories i.e. Controllers, Services, and DAOs.

Q.How is bean name defined for auto-scan ?
Ans.By default, the bean names are derived from class names with a lowercase initial character.
Example:

```
	@Component
	public class CustomerLogging{	------->Bean Name=customerLogging
		//rest of the code
	}

	@Repository
	public class CustomerRepositoryImpl implements CustomerRepository {
	Bean Name=customerRepositoryImpl
		//rest of the code
	}
```

IF we want to give bean name then we can use "value" attribute

```
	@Repository(value="customerRepository") or
@Repository("customerRepository")
	public class CustomerRepositoryImpl implements CustomerRepository
{
		//rest of the code
	}
```

CodeExample:

```
@Service("customerService")
public class CustomerServiceImpl implements CustomerService {
	@Value("10")
	private int count;

	public String fetchCustomer() {
		return " The no of customers fetched are : " + count;
	}
}

@Configuration
@ComponentScan(basePackages="com.infy")
public class SpringConfiguration {
}

public class Client {
	public static void main(String[] args) {
		CustomerServiceImpl service = null;
		AbstractApplicationContext context = new
AnnotationConfigApplicationContext(SpringConfiguration.class);
		service = (CustomerServiceImpl)
context.getBean("customerService");
		System.out.println(service.fetchCustomer()); //-> The no of
customers fetched are 10
		context.close();
	}
```

}

Q.Why Spring Boot ?
Ans.Reduces time for configuration so that developer can focus on Business Logic.Also no need to add dependencies manually .Initially, if the wrong version of dependencies is selected then it will be an uphill task to solve this problem.All these things are resolved.

Spring Boot is a framework built on top of the Spring framework that helps the developers to build Spring-based applications very quickly and easily.Spring Boot features:
1. Spring Boot is an opinionated framework :It uses its own embedded tomcat to run application
2. Spring Boot is customizable:Though Spring Boot has its defaults, you can easily customize it at any time during your development based on your needs. For example, if you prefer log4j for logging over Spring Boot built-in logging support then you can easily make a dependency change in your pom.xml file to replace the default logger with log4j dependencies.

Q.what are the main Spring Boot features ?
Ans.1.Starter Dependencies
2.Automatic Configuration
3.Spring Boot Actuator
4.Easy-to-use Embedded Servlet Container Support

Q.What are dependencies available for project created without adding additional dependencies?
Ans.spring-boot-starter : This is the core starter that includes support for auto-configuration, logging.
spring-boot-starter-test : This starter provides support for testing Spring Boot applications using libraries such as JUnit, Hamcrest, and Mockito.

Q.What are Spring Boot Runners?
Ans.If we want to perform some operations Just after "application starts" then for this Spring Boot provides the following two interfaces:
1.CommandLineRunner interface
2.ApplicationRunner interface

CommandLineRunner has run method which gets automatically called once spring boot has "loaded application context"

Q.How can we use data present in properties file ?
Ans. By default we can use data of "application.properties" file.

Example:
1.In application.properties file
message= Welcome Spring
2.@Autowired
Environment env;

3.env.getProperty("message")

If we want to use some other file .For example,
InfyTelmessage.properties.
Example:
1.InfyTelmessage.properties
message=Welcome To InfyTel

2.@SpringBootApplication
@PropertySource("classpath:InfyTelmessage.properties")****FOCUS******
public class DemoSpringBootApplication {
public static void main(String[] args) throws Exception {
SpringApplication.run(SwitchPartyApplication.class, args);
}
}

Q.Explain Below piece of code ?
@SpringBootApplication
public class DemoSpringBootApplication {
public static void main(String[] args) {
SpringApplication.run(DemoSpringBootApplication.class, args);
}
}
Ans.@SpringBootApplication is combination of
1.@EnableAutoConfiguration – This annotation enables auto-configuration for the Spring boot application which automatically configures our application based on the "DEPENDENCIES" that you have added.
2.@ComponentScan – All application components which are annotated with @Component, @Service, @Repository, or @Controller are automatically registered as Spring Beans. These beans can be injected by using @Autowired annotation.
@Configuration – This enables Java based configurations for Spring boot application.

The class that is annotated with @SpringBootApplication will be considered as the main class.it starts application by by invoking the SpringApplication.run() method

Q.Explain SpringBootApplication- scanBasePackages?
Ans.By default, SpringApplication scans the configuration class package and all it’s sub-packages. So if our SpringBootApplication class is in "com.eta" package, then it won’t scan com.infy.service or com.infy.repository package. We can fix this situation using the SpringBootApplication scanBasePackages property.

package com.eta;
@SpringBootApplication(scanBasePackages={"com.infy.service","com.infy.repository"})
public class DemoSpringBootApplication {
public static void main(String[] args) {
SpringApplication.run(DemoSpringBootApplication.class, args);
}
}

Q.how can we get ApplicationContext object in spring boot project ?
Ans.ApplicationContext
context=SpringApplication.run(SwitchPartyApplication.class, args);

Q.Define autowiring ?

Ans.if one bean class is dependent on another bean class ,Then Spring IoC container to inject the dependencies into dependent bean classes without been defined in your configuration class  This is called as autowiring.

OR

It is used to inject one bean into another bean
Annotation used is @autowired

Q.Where can we use @autowired?
Ans.It can be applied to attributes, constructors, setter methods of a bean class.

Q.Define @value ?
Ans.Autowiring is done only for dependencies to other beans. It doesn't work for properties such as primitive data types, String, Enum, etc.
For such properties, you can use the @Value annotation.

OR

It is an annotation used to set values for primitive data types when bean loads
Example:

```
public class CustomerDTO {
      @Value("1234567891")
      long phoneNo;
      @Value("Jack")
      String name;
      @Value("Jack@xyz.com")
      String email;
      @Value("ANZ")
      String address;
}
```

Q.Explain autowiring for Setter Injection ?
Ans.

```
package com.infy.service;
public class CustomerServiceImpl implements CustomerService {

      private CustomerRepository customerRepository;
      @Autowired
      public void setCustomerRepository(CustomerRepository
customerRepository) {
            this.customerRepository = customerRepository;
      }
      --------
}
```

Q.Explain autowiring for Constructor Injection ?
Ans.

```
package com.infy.service;
public class CustomerServiceImpl implements CustomerService {

      private CustomerRepository customerRepository;
      @Autowired
      public CustomerServiceImpl(CustomerRepository customerRepository) {
            this.customerRepository = customerRepository;
      }
}
```

Q.Q.Explain autowiring for Instance variables/attributes ?
Ans.

```
package com.infy.service;
public class CustomerServiceImpl {

      @Autowired
      private CustomerRepository customerRepository;

}
```

Q.what if there are mutiple beans available for same type ?
Ans.If more than one beans of the same type are available in the container, then the framework throws an exception indicating that more than one bean is available for autowiring. To handle this @Qualifier annotation is used as follows:

```
package com.infy.service;
public class CustomerServiceImpl {
            @Autowired *****FOCUS******//y b use hua h @Qualifier k sath
         @Qualifier("custRepo")******FOCUS****** dekho y kse use kia h
      private CustomerRepository customerRepository;

}
```

Q.what is BeanScope?
Ans.Bean scope refers to life cycle of bean i.e  how long does bean live how many instances will be shared
Default scope :Singleton

Q.What does Singleton means ?
Ans.har bean request k liye ....Only one instance of bean is created and shared among all .Default scope is also singleton .

Example1:

```
@Service("customerService")
@Scope("singleton") OR @Scope("prototype") ********FOCUS******
public class CustomerServiceImpl implements CustomerService {.........}
```

Example2:

```
            CustomerService
cs1=context.getBean("customerService",CustomerService.class);
            CustomerService
cs2=context.getBean("customerService",CustomerService.class);
            System.out.println(cs1==cs2);//true bcoz by default scope
singleton h
```

Q.What is logging?
Ans.Logging is the process of tracking the execution of a program.

Q.Why do we Require Logging ?
Ans.Recording unusual circumstances or errors that may be happening in the program
      Getting the info about what's going in the application.
      It helps in quick problem diagnosis and debugging,

Q.ALAG alag logging levels me kya farak h?

Ans.Alag alag logging levels specify "Severity of an event to be logged".For eg. TRACE can be used during development and
ERROR during deployment

| | |
|---|---|
| ALL | For all levels |
| TRACE | Informational Events |
| DEBUG | Informational that would be useful for debugging application |
| INFO | Information that highlights progress of application |
| WARN | Harmuful si tuations |
| ERROR | errors that permit application to continue running |
| FATAL | errors that stop the application |
| OFF | To disable all levels |

Q.We have not defined anything for logging initially but still we can see INFO level logging messages in console while running application?
Ans.The reason is Spring Boot's default support for logging. The spring-boot-starter dependency includes spring-boot-starter-logging dependency, which configures logging via "Logback" to log to the console at the INFO level.
Spring Boot uses Commons Logging API with default configurations for
Java Util Logging,
Log4j 2,
and Logback implementation.
Among these implementations, Logback configuration will be enabled by default.

Methods available are :
void debug(Object msg)
void error(Object msg)
void fatal(Object msg)
void info(Object msg)
void warn (Object msg)
void trace (Object msg)

Q.Explain 2021-06-26 15:43:44.327 INFO 4332 --- [ main] com.Infosys.SwitchPartyApplication : No active profile set, falling back to default profiles: default ?
Ans.Date and Time :2021-06-26 15:43:44.327
Log level:INFO
Process id:4332
Separator:---
Thread Name:[ main]
Logger Name:com.Infosys.SwitchPartyApplication
Log message: No active profile set, falling back to default profiles: default ?
---