Question 135: How to configure the port number of a SpringBoot application?

Answer: By default, the embedded server starts on port 8080, however you can change it by defining a property in application.properties file, like:

server.port=8082

You can also pass it as a vm argument:

java –jar C:\temp\app.jar –server.port=8082

or

java –jar –Dserver.port=8082 C:\temp\app.jar

Question 136: Which jar builds our springboot application automatically whenever we change some code just like a node.js application?

Answer: Devtools dependency, just add the below maven dependency in your pom.xml,

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
</dependency>
```

Question 137: What default embedded server is given in spring boot web starter and how we can change the default embedded server to the one of our choice

Answer: The default embedded server is Tomcat, that comes with Spring boot web starter, if you want to change this, then use <exclusion> tag in web starter and add a separate dependency of the server that you want.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <exclusions>
        <exclusion>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-tomcat</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

When you add the exclusion tag and save the pom.xml, you will see that the tomcat dependency will be removed from Maven Dependencies, then you can add another server's dependency like the one below:

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

Question 138: Where should we put our html and javascript files in a spring boot application?

Answer: If you are adding html and javascript files, then you should put them in *static folder* in **src/main/resources/**. You should also make separate folders for html, css, javascript files.

Question 139: What are the different stereotype annotations?

Answer: @Component, @Controller, @Service and @Repository annotations are called stereotype annotations and they are present in *org.springframework.stereotype* package.

Question 140: Difference between @Component, @Controller, @Service, @Repository annotations?

Answer:

@Component: it is a general purpose stereotype annotation which indicates that the class annotated with it, is a spring managed component.

@Controller, @Service and @Repository are special types of @Component, these 3 themselves are annotated with @Component, see below

```
@Target(value=TYPE)
@Retention(value=RUNTIME)
@Documented
@Component
public @interface Controller
```

```
@Target(value=TYPE)
@Retention(value=RUNTIME)
@Documented
@Component
public @interface Service
```

```
@Target(value=TYPE)
@Retention(value=RUNTIME)
@Documented
@Component
public @interface Repository
```

So, the classes annotated with these annotations gets picked up in Component scanning and they are managed by Spring.

@Controller: the classes annotated with @Controller will act as Spring MVC controllers. DispatcherServlet looks for @RequestMapping in classes that are annotated with @Controller. That means you cannot replace @Controller with @Component, if you just replace it with @Component then yes it will be managed by Spring but it will not be able to handle the requests.

(Note: if a class is registered with Spring using @Component, then @RequestMapping annotations within class can be picked up, if the class itself is annotated with @RequestMapping)

@Service: the service layer classes that contain the business logic should be annotated with @Service. Apart from the fact that it is used to indicate that the class contains business logic, there is no special meaning to this annotation, however it is possible that Spring may add some additional feature to @Service in future, so it is always good idea to follow the convention.

@Repository: the classes annotated with this annotation defines data repositories. It is used in DAO layer classes. @Repository has one special feature that it catches platform specific exceptions and re-throw them as one of the Spring's unified unchecked exception i.e. *DataAccessException*.

Question 141: Difference between @Controller and @RestController annotation

Answer: The differences are:

- @Controller annotation is used to mark a class as Spring MVC controller where the response is a view name which will display the Model object prepared by controller, whereas @RestController annotation is a specialization of @Controller and it is used in RESTful web services where the response is usually JSON/XML.
- @RestController is made up of 2 annotations, @Controller and @ResponseBody. @ResponseBody annotation is used to attach the generated output directly into the body of http response.
- @Controller can be used with @ResponseBody which will have same effect as @RestController. @ResponseBody annotation can be used at the class level or at the individual methods also. When it is used at the method level, Spring will use HTTP Message Converters to convert the return value to HTTP response body (serialize the object to response body).

Question 142: What is @RequestMapping and @RequestParam annotation?

Answer: @RequestMapping annotation maps the incoming HTTP request to the handler method that will serve this request, we can apply @RequestMapping on a class as well as a method. When used on a class, it maps a specific request path or pattern onto a controller, then the method level annotation will make the mappings more specific to handler methods.

@RequestParam annotation is used to bind a web request parameter to the parameter of handler method

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/home")
public class DemoController {

    @RequestMapping(value="/getById")
    public String getById(@RequestParam(value = "id") String requestId) {
        return "dummy response";
    }
}
```

In the above @RequestParam example, we have defined a value parameter that just denotes that the incoming request parameter is by the name 'id' which will be mapped to the variable name 'requestId', you can define @RequestParam without this value attribute also.

By default, all requests are assumed to be of type HTTP GET, however we can specify the request type by using 'method' attribute of @RequestMapping annotation, like

```
@RequestMapping(method = RequestMethod.POST)
public String post() {
    return "Post request";
}

@RequestMapping(method = RequestMethod.PUT)
public String put() {
    return "Put request";
}
```

```
@RequestMapping(method = RequestMethod.PATCH)
public String patch() {
    return "Patch request";
}

@RequestMapping(method = RequestMethod.DELETE)
public String delete() {
    return "Delete request";
}
```

Question 143: How to define a Get or Post endpoint?

Answer: You can either use the @RequestMapping and defining its 'method' attribute as RequestMethod.GET, RequestMethod.POST or you can also use the shortcuts like @GetMapping and @PostMapping annotations.

```
@GetMapping("/getByName/{name}")
public String getMapping(@PathVariable String name) {
    return "Get";
}

@PostMapping("/saveUser")
public String postMapping() {
    return "Post";
}
```

There are similar shortcut annotations present for PUT, PATCH, DELETE requests too.

Question 144: Which annotation is used for binding the incoming json request to the defined pojo class?

Answer: @RequestBody annotation is used to bind the incoming json request to the pojo class,

```
@RequestMapping(value = "/createUser", method = RequestMethod.POST)
public String createUser(@RequestBody User user) {
    return "Created user " + user.getName();
}
```

Behind the scenes, Spring uses Jackson library (that comes with spring-boot-starter-web) to map the json request to the pojo class. *MappingJackson2HttpMessageConverter* is used when the incoming request is of content-type application/json.

Question 145: What is @Qualifier annotation?

Answer: Let's consider an example to understand @Qualifier annotation better. Suppose we have an interface called Shape and there are 2 classes Rectangle and Circle that are implementing this interface. We are autowiring our Shape interface in our controller class using @Autowired, now here a conflict will happen, because there are 2 beans of the same type.

```
public interface Shape {


}
```

```java
@Service
public class Rectangle implements Shape {

}
@Service
public class Circle implements Shape {

}
@RestController
public class ShapeController {

    @Autowired
    Shape shape;
```

When you try to start your application, you will get:

```
***************************
APPLICATION FAILED TO START
***************************

Description:

Field shape in com.example.demo.controller.ShapeController required a single bean, but 2 were found:
    - circle: defined in file [C:\demo\target\classes\com\example\demo\service\Circle.class]
    - rectangle: defined in file [C:\demo\target\classes\com\example\demo\service\Rectangle.class]
```

Now, to resolve this you can give names to your Rectangle and Circle class, like:

```java
@Service("rectangle")
public class Rectangle implements Shape {

}
@Service("circle")
public class Circle implements Shape {

}
```

And you will use @Qualifier annotation to specify which bean should be autowired, like:

```java
@RestController
public class ShapeController {

    @Autowired
    @Qualifier("circle")
    Shape shape;
```

Now, Spring will not get confused as to what bean it has to autowire.

NOTE, you can also use @Qualifier annotation to give names to your Rectangle and Circle classes, like

```
@Service
@Qualifier("rectangle")
public class Rectangle implements Shape {

}
```

Question 146: What is @Transactional annotation?

Answer: Spring provides Declarative Transaction Management via @Transactional annotation. When a method is applied with @Transactional, then it will execute inside a database transaction. @Transactional annotation can be applied at the class level also, in that case, all methods of that class will be executed inside a database transaction.

How @Transactional works:

When @Transactional annotation is detected by Spring, then it creates a proxy object around the actual bean object. So, whenever the method annotated with @Transactional is called, the request first comes to the proxy object and this proxy object invokes the same method on the target bean. These proxy objects can be supplied with interceptors. Spring creates a TransactionInterceptor and passes it to the generated proxy object. So, when the @Transactional annotated method is called, it gets called on the proxy object first, which in turn invokes the TransactionInterceptor that begins a transaction. Then the proxy object calls the actual method of the target bean. When the method finishes, the TransactionInterceptor commits/rollbacks the transaction.

One thing to remember here is that the Spring wraps the bean in the proxy, the bean has no knowledge of it. So, only the external calls go through the proxy. As for the internal calls (@Transactional method calling the same bean method), they are called using 'this'.

Using @Transactional annotation, the transaction's propagation and isolation can be set directly, like:

```
@Transactional(propagation = Propagation.REQUIRES_NEW,
               isolation = Isolation.READ_UNCOMMITTED,
               rollbackFor = Exception.class)
public String process() {
    return "Success";
}
```

Also, you can specify a 'rollbackFor' attribute and specify which exception types must cause a transaction rollback (a transaction with Runtime exceptions and errors are by default rolled back).

If your process() method is calling another bean method, then you can also annotate that method with @Transactional and set the propagation level to decide whether this method should execute in the same transaction or it requires a new transaction.

Question 147: What is @ControllerAdvice annotation?

Answer: @ControllerAdvice annotation is used to intercept and handle the exceptions thrown by the controllers across the application, so it is a global exception handler. You can also specify @ControllerAdvice for a specific package,

```
@ControllerAdvice(basePackages="com.example.demo.controller")
public class Test {
```

Or a specific controller,

```
@ControllerAdvice(assignableTypes=DemoController.class)
public class Test {
```

Or even a specific annotation,

```
@ControllerAdvice(annotations=RestController.class)
public class Test {
```

@ExceptionHandler annotation is used to handle specific exceptions thrown by controllers, like,

```
@ControllerAdvice
public class Test {

    @ExceptionHandler(SQLException.class)
    public String handleSQLException() {
        return null;
    }

    @ExceptionHandler(UserNotFoundException.class)
    public String handleUserNotFoundException() {
        return null;
    }

}
```

Here, we have defined a global exception handler using @ControllerAdvice. If a SQLException gets thrown from a controller, then handleSQLException() method will be called. In this method, you can customize the exception and send a particular error page/error code. Also, custom exceptions can be handled.

If you don't want to create a global exception handler, then you can also define some @ExceptionHandler methods in a particular controller itself.

Question 148: What is @Bean annotation?

Answer: @Bean annotation is used when you want to explicitly declare and register a bean into application context, so that it will be managed by Spring.

Some points to remember:

- When using @Bean, you have the control over the bean creation logic.
- @Bean is a method level annotation, the body of the method contains the logic for creating the bean instance and this method returns the instance which will be registered in the spring application context.
- Using @Bean, you can register the classes from 3rd party libraries into the application context
- @Bean annotation is usually declared in configuration classes.

Question 149: Difference between @Component and @Bean

Answer: The differences are:

- @Component auto-detects and configures the beans using classpath scanning, whereas @Bean explicitly declares a single bean rather than letting Spring do it automatically
- @Component is a class level annotation, whereas @Bean is a method level annotation
- @Component has different specializations called stereotype annotations like @Controller, @Service and @Repository, whereas @Bean has no specializations
- @Bean lets you create and configure beans exactly how you choose it to be, whereas in @Component, Spring has the control
- @Bean lets you configure classes from 3rd party libraries where you are not the owner of the source code, but you can't use @Component in this case

Question 150: How to do profiling in a SpringBoot application

Answer: Every application has many environments like DEV, STAGE, UAT, Pre-PROD, PROD. And all these environments have different configurations like the database properties, port number etc. In Spring boot, we define all properties in application.properties file. But if we want to maintain different profiles, then we can have multiple application.properties file for each environment and at the runtime, we can specify which profile should be active.

The profiles can be created using the syntax, ***application-{profile_name}.properties***

For example,

application-dev.properties for dev environment specific configs

application-uat.properties for uat environment specific configs

application-prod.properties for prod environment specific configs

How to activate a particular profile:

There are many ways to activate a particular profile, one of them is by defining a property in our master application.properties file, like

spring.profiles.active=uat

We can also set a particular profile to be active by passing it as a VM argument, Maven settings, JVM system parameters, environment variables. In xml, we use <beans profile="uat"> to activate a profile.

There is one @Profile annotation which when applied to a component tells that this component will be created only when a particular profile is set.

Question 151: What is RestTemplate?

Answer: RestTemplate is used to consume other REST services programmatically. It can also bind the api response to the custom domain classes. You must have used/seen the HttpURLConnection to consume other services, it contains a lot of boilerplate code, like configuring the HTTP request, executing it and then converting the HTTP response to a Java object. But when using RestTemplate, all these things happen in the background.

```
public String getUserNameById() {
    User user = new RestTemplate().getForObject("http://localhost:8080/getUser/1", User.class);
    return user.getName();
}
```

In this example, we are consuming a GET web service and converting the response to the object of User class.

Similarly, there are other methods to consume POST, DELETE web services like exchange() and delete() respectively.

Question 152: What is Spring Data JPA?

Answer: Spring Data is part of Spring framework. It provides an abstraction to significantly reduce the amount of boilerplate code required to implement data access layers. It does this by providing Repositories. Repositories are just interfaces that do not have any implementation class and provide various database operations like save, delete etc. We have several repositories to choose from, CRUDRepository, JPARepository, PagingAndSortingRepository.

If you are thinking how Spring Data JPA saves us from writing boilerplate code, then consider an example, you have an Employee class and you are writing its Dao interface and implementation class for performing some CRUD operations. You will also have other classes and similarly, you will write CRUD operations logic for these classes as well. So, there is a lot of boilerplate code here. Spring Data JPA takes care of this, and provides you with a Repository interface which have all the common DAO operations. We just have to extend these Repositories and Spring Data JPA will provide the DAO implementation at runtime.

Spring Data JPA can also generate JPA queries on our behalf, we just have to follow certain method naming conventions and the database query will be automatically generated for us.

For example, let's say we have a table named Employee with id, name and age columns. If you want to write a query that will give the Employee object by name, then you can use Spring Data JPA, like:

public EmployeeEntity findByName(String name);

Spring Data JPA will transform this method into:

select * from Employee where name='passed_name_value';

Question 153: What is the difference between JPARepository, CRUDRepository, PagingAndSortingRepository and which one you have used in your applications?

Answer: JPARepository extends the PagingAndSortingRepository which in turn extends the CRUDRepository. So if you are using JPARepository, you can use all the methods of other two also.

CRUDRepository – this repo mainly provides methods related to CRUD operations

PagingAndSortingRepository – this repo extends the CRUDRepository and provides methods to do pagination and sorting of records

JPARepository – this repo extends the PagingAndSortingRepository and provides JPA related methods like flushing the persistence context and deleting records in batches

You can extend any of these repository and make a customized repo to have methods according to your need.

Question 154: What is Spring AOP?

Answer: Spring AOP (Aspect Oriented Programming) is one of the key components of Spring framework.

Consider an example to understand the AOP better, let's say we have 100 methods and you have to maintain logs in these methods, like what is the method name and what is the result of each method. One way is to go in each method and write logger statements. What if this requirement changes and you don't need this logger anymore then you will again go in each method and remove this logger. One thing is clear from this example that Logging is a cross-cutting concern.

A cross-cutting concern is a concern that can affect the entire application and should be centralized in one location as much as possible. Few examples of cross-cutting concerns are, transaction management, logging, authentication etc.

AOP helps you to refactor the different necessary repeating codes into different modules. By doing this, the cluttering of code can be removed and the focus can be applied on the actual business logic. AOP provides modularity but here the key unit of modularity is Aspect. Using AOP, you can add extra functionality before or after a method execution.

AOP terminology:

Aspect: Aspect is a class that implements the application concerns that cuts across multiple classes, such as transaction management and logging. Aspects can be a normal class configured through Spring XML configuration or we can use @Aspect annotation.

Join Point: a point during the execution of a program, such as the execution of a method or the handling of an exception. In Spring AOP, a join point always represents a method execution.

Advice: advices are actions that are taken for a particular join point. There are different types of advices, like, before, after and around advice.

Pointcut: pointcut is an expression that is matched with join points to determine whether advice needs to be applied or not.

Target Object: objects on which advices are applied by one or more aspects. Since Spring AOP is implemented using runtime proxies, this object will always be a proxied object.

AOP Proxy: an object created by the AOP framework in order to implement the aspect contracts (advise method executions and so on). In the Spring Framework, an AOP proxy will be a JDK dynamic proxy or a CGLIB proxy.

Weaving: It is the process of linking aspects with other objects to create the advised proxy objects. This can be done at compile time, load time or at runtime. Spring AOP performs weaving at the runtime.

AOP Advice types:

Before advice: these advices run before the execution of join point methods. @Before annotation is used to mark a method as before advice.

After returning advice: Advice to be executed after a join point completes normally: for example, if a method returns without throwing an exception. @AfterReturning annotation is used to mark a method as after returning advice.

After throwing advice: Advice to be executed if a method exits by throwing an exception. @AfterThrowing annotation marks a method as after throwing advice.

After (finally) advice: An advice that gets executed after the join point method finishes executing, whether normally or by throwing an exception. @After annotation is used to create an after advice.

Around advice: this advice surrounds a join point. Around advice can perform custom behavior before and after the method invocation. It is also responsible for choosing whether to proceed to the join point or to shortcut the advised method execution by returning its own return value or throwing an exception. @Around annotation is used create around advice methods.

Question 155: Have you used Spring Security in your application

Answer: If you are giving a Spring/SpringBoot interview, then this question will definitely be asked that how did you secure your application APIs? Every application uses some type of security to protect the unwanted access, so if you are working in a project and even if you have not implemented the security part yourself, then I suggest, you should try to understand how it is implemented and read more about it. This will be better than not answering the question. You can always say you worked on it as a team or you have implemented it in some of your personal projects

Question 156: What do you know about Spring Batch framework?

Answer: Spring batch framework can be used when a task requires batch processing, for example, generating financial reports at end of day or month. Batch processing includes tasks like reading and writing to files, transforming data, reading and writing to databases etc. These steps are often chained together and can also be executed sequentially or in parallel. If an error occurs in a batch job step, then it can be found out that in which step, the error has occurred and it is also possible to resume that job execution from that failed step.

Question 157: Difference between SOAP and REST

Answer: The differences are:

- SOAP stands for Simple Object Access Protocol and REST stands for Representational State Transfer
- SOAP is a protocol whereas REST is an architectural style
- SOAP cannot use REST because it is a protocol whereas REST can use SOAP web services as the underlying protocol, because it is just an architectural pattern
- SOAP uses service interfaces to expose its functionality to client applications whereas REST uses URI to expose its functionality
- SOAP defines standards that needs to be strictly followed for communication to happen between client and server whereas REST does not follow any strict standard

- SOAP requires more bandwidth and resources than REST because SOAP messages contain a lot of information whereas REST requires less bandwidth than SOAP because REST messages mostly just contains a simple JSON message
- SOAP only works with XML format whereas REST allows different data formats like Plain text, HTML, XML, JSON etc.
- SOAP defines its own security whereas REST inherits the security

Question 158: What is Restful api?

Answer: For an api to be Restful, it has to satisfy 6 constraints, they are:

Client-Server: This constraint operates on the concept that the client and the server should be separate from each other and allowed to evolve individually.

Stateless: REST is stateless, it means the client request should contain all the information that is needed for a server to respond.

Cacheable: Cache constraint requires that response returned from a server should be labelled as cacheable or non-cacheable either implicitly or explicitly. If the response is defined as Cacheable then the client can re-use the response data for future requests because a stateless API can increase request overhead by handling large loads of incoming and outgoing calls.

Uniform Interface: The key to decoupling the client from server is having a uniform interface that allows independent evolution of the application without having the application's services, or models and actions, tightly coupled to the API layer itself.

Layered System: REST APIs have different layers of their architecture working together to build a hierarchy that helps create a more scalable and modular application.

Code on Demand (Optional): this is an Optional constraint. It allows servers to temporarily extend the functionality of a client by downloading and executing code in the form of applets or scripts.

Question 159: What is a stateless object?

Answer: An instance of a class without any instance fields is a stateless object. The class can have fields but they can only be constants, *static final*.

Question 160: What is the difference between Web Server and Application Server?

Answer: The differences are:

- Web server provides environment to run only web related applications, whereas Application Server provides environment to run Java J2EE applications (enterprise applications)
- Web server is used to deliver static contents like static html pages, whereas through Application server, you can deliver dynamic content
- Web servers can be used for Servlets, JSPs whereas Application servers can be used for Servlets, JSPs, EJBs, JMS etc. (Application servers have an internal web server inside it to serve web applications)
- Web servers support HTTP protocol, whereas Application servers support HTTP as well as RPC/RMI protocols
- Web server consume less resources than Application servers
- Tomcat, Jetty are examples of Web Servers, whereas GlassFish, JBoss, WebLogic, WebSphere are some examples of Application servers

Question 161: What do you know about CommandLineRunner and ApplicationRunner?

Answer: SpringBoot provides us with two interfaces, CommandLineRunner and ApplicationRunner. Both these runners have a run() method, which gets executed just after the application context is created and before SpringBoot application startup.

You just have to register these runners as beans in application context. Then Spring will automatically pick them up.

Both of these interfaces provide the same functionality and the only difference between them is CommandLineRunner.run() method accepts String arr[], whereas ApplicationRunner.run() method accepts ApplicationArguments as argument.

Multiple runners can be defined and we can order them as well, either by extending the interface org.springframework.core.Ordered or via the @Order annotation.

Knowing these topics well should be enough to crack a Spring/SpringBoot interview, if you want to really impress the interviewer, you can prepare the below topics as well:

- Spring Cloud, how different environment properties are stored in Git
- Eureka Naming Server
- Zuul
- Zipkin
- Hysterix

All the above topics are advanced but I will explain about them a little, so you can go out and read more about it.

Question 162: What do you know about Eureka Naming Server?

Answer:

To know about the need of Eureka Naming Server and what it does, let's consider an example. Suppose you have 5 micro-services which are experiencing heavy traffic and you want to deploy multiple instances of these 5 micro-services and use a load balancer to distribute the traffic among these instances. Now, when you create new instances of your micro-service, you have to configure these in your load balancer, so that load balancer can distribute traffic properly. Now, when your network traffic will reduce then you will most likely want to remove some instances of your micro-service, means you will have to remove the configuration from your load balancer. I think, you see the problem here.

This manual work that you are doing can be avoided by using Eureka naming server. Whenever a new service instance is being created/deleted, it will first register/de-register itself to the Eureka naming server. Then you can simply configure a ***Ribbon client*** (Load Balancer) which will talk with Eureka Naming server to know about the currently running instances of your service and Ribbon will properly distribute the load between them. Also, if one service, serviceA wants to talk with another service, serviceB then also Eureka Naming server will be used to know about the currently running instances of serviceB.

You can configure Eureka Naming Server and Ribbon client in SpringBoot very easily.

Question 163: What do you know about Zuul?

Answer:

Zuul is an API gateway server. It handles all the requests that are coming to your application. As it handles all the requests, you can implement some common functionalities of your micro-services as part of Zuul server like Security, Monitoring etc. You can monitor the incoming traffic to gain some insights and also provide authentication at a single place rather than repeating it in your services. Using Zuul, you can dynamically route the incoming requests to the respective micro-services. So, the client doesn't have to know about the internal architecture of all the services, it will only call the Zuul server and Zuul will internally route the request.

Question 164: What do you know about Zipkin?

Answer:

To understand Zipkin use-case, let's consider an example. Suppose you have a chain of 50 micro-services where first micro-service is calling the second and second calling third and so on. Now, if there is an error in, say 35th micro-service, then how you will be able to identify your request that you made to the first micro-service, from all the logs that gets generated in all 35 micro-services. I know this is an extreme example

Zipkin helps in distributed tracing, especially in a micro-service architecture. It assigns an 'id' to each request and gives you a dashboard, where you can see the complete request and a lot more details, like the entire call-chain, how much time one micro-service took and which service failed etc.

Question 165: What do you know about Hysterix?

Answer:

Hysterix is a library that makes our micro-service, fault-tolerant. Suppose, you have a chain of 10 micro-services calling each other and the 6th one fails for some reason, then your application will stop working until the failed micro-service is fixed.

You can use Hysterix here and provide a fallback method in case of a service failure.

```
@GetMapping("/getByName/{name}")
@HystrixCommand(fallbackMethod = "handlerMethod")
public String getMapping(@PathVariable String name) {
    return "Get";
}

public String handlerMethod() {
    return "Service is down";
}
```

If the GET service is getting failed, then the fallback method will be executed.

JPA / Hibernate

Question 166: What is JPA?

Answer: JPA stands for Java Persistence API. It is a specification which gives a standard API for accessing databases within java applications. As JPA is just a specification, it does not perform any operation by itself. It requires an implementation, there are many JPA implementations available like Hibernate, iBatis, TopLink, EclipseLink etc. Hibernate ORM is the most popular implementation of JPA.

Question 167: What is Hibernate?

Answer: Hibernate is an Object Relational Mapping tool (ORM tool), that maps the Java objects to the database tables and vice-versa.

Some points to remember:

- Hibernate framework provides the facility to create database tables automatically
- Hibernate framework provides us object-oriented version of SQL known as HQL (Hibernate Query Language). It generates the database independent queries. So, even if our database gets changed, we don't have to change our SQL queries according to the new database
- Using Hibernate, we can define relationships between our Entities (tables), that makes it easy to fetch data from multiple tables
- Hibernate supports Caching, that improves the performance of our application
- Using Hibernate, we can generate the Primary key of our tables automatically

Question 168: Difference between JPA and Hibernate

Answer: JPA is just a specification i.e. it defines a set of concepts that can be implemented by any tool or framework, and Hibernate is one of the implementation of JPA.

Question 169: What is @Entity?

Answer: @Entity annotation defines that a class can be mapped to a database table. The class fields will be mapped to the columns of the table.

Question 170: How to give a name to a table in JPA?

Answer: @Table annotation can be used to give name to a table

```
import javax.persistence.Entity;
import javax.persistence.Table;

@Entity
@Table(name="USER")
public class User {
```

Question 171: What is @Id, @GeneratedValue?

Answer: @Id annotation defines the primary key of a table and @GeneratedValue annotation is used to specify the primary key generation strategy to use. If the strategy is not specified, the default strategy AUTO will be used.

```
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
private int userId;
```

Question 172: How to use a custom database sequence in Hibernate to generate primary key values?

Answer: JPA specification provides a set of annotations to define the primary key generation strategy. To use a custom sequence, we have to set the *GenerationType* to SEQUENCE in @GeneratedValue annotation, this tells Hibernate to use a database sequence to generate the primary key value. If we don't provide any other information, Hibernate will use its default sequence.

To define the name and schema of the database sequence, there is an annotation @SequenceGenerator.

See the code snippet for doing all this:

```
@Id
@GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "userSeq_generator")
@SequenceGenerator(name = "userSeq_generator", sequenceName = "user_seq")
```

If you persist a new entity, then Hibernate will use 'user_seq'.

Question 173: How to give names to the columns of a JPA Entity

Answer: @Column annotation can be used to give names to a column

```
@Column(name="address")
private String userAddress;
```

Question 174: How to define a @OneToMany relationship between entities

Answer: Here the interviewer will ask you about the syntax, so to define a One to Many relationship, consider a *class Cart* that can have multiple items represented by *class Item*. So, one Cart -> many Items.

Cart Entity:

```
@Entity
public class Cart {

    @OneToMany(mappedBy = "cart")
    private List<Item> items;
```

In mappedBy attribute, the field name of Item entity is passed, see below,

Item Entity:

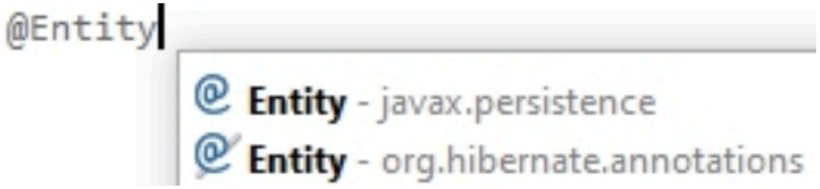
```
@Entity
public class Item {

    @ManyToOne
    @JoinColumn(name = "CART_ID")
    private Cart cart;
```

@JoinColumn annotation is used to define the name of foreign key column that represents the entity association.

Question 175: Why annotations should be imported from JPA and not from Hibernate?

Answer: If you see @Entity annotation, it is present in *javax.persistence* package and also in *org.hibernate.annotations* package:



You should choose the one from *javax.persistence* package, because if you choose the Hibernate one, then in future, if by any chance, you have to remove Hibernate and use some other JPA implementation, like iBatis, then you will have to change your code (imports). As you remember, JPA is just a specification, like an interface. You can plug any of its implementations as long as you use the imports from *javax.persistence* package.

Question 176: What is the difference between get() and load() method of Hibernate Session?

Answer: Hibernate Session class provides two methods to access object, session.get() and session.load()

The differences are:

- get() method involves a database hit, if the object does not exist in Session cache and it returns a fully initialized object which may involve several database calls, whereas load() method returns a proxy object and it only hit the database if any method other than getId() is called on the entity object
- load() method results in slightly better performance as it can return a proxy object, it will only hit the database when a non-identifier getter method is called, whereas get() method returns a fully initialized object when it does not exist in Session cache which may involve multiple database calls based on entity relationships
- get() method returns null if the object is not found in the cache as well as the database whereas load() method will throw ObjectNotFoundException but never return null
- If you are not sure whether the object exists or not, then use get() as it will return null but if you are sure that the object exists, then use load() method as it is lazily initialized

Question 177: What is the difference between save(), saveOrUpdate() and persist() method of Hibernate Session?

Answer: Hibernate Session class provides various methods to save an object into the database, like save(), saveOrUpdate() and persist()

The difference between save() and saveOrUpdate() method is that save() method saves the record into the database by INSERT query, generates a new identifier and returns the Serializable identifier back, while saveOrUpdate() method either INSERT the record or UPDATE the record if it already exists, so it involves extra processing to find whether the record already exists in the table or not.

Similar to save(), persist() method is also used to save the record into the database table.

The differences between save() and persist() are:

- Return type of persist() method is void while return type of save() method is Serializable object
- Both persist() and save() methods makes a transient instance persistent. But persist() method does not guarantee that the identifier value will be assigned to the persistent instance immediately, the assignment might happen at flush time
- Both behave differently when they are executed outside the transaction boundaries. persist() method ensures that it will not execute an INSERT when it is called outside of a transaction boundary whereas save() method does not guarantee this, it returns an identifier and if an INSERT query has to be executed to get the identifier then this INSERT happens immediately and it does not matter if the save() is called inside or outside of a transaction
- persist() method is useful in long-running conversation with an extended Session context because it does not execute an INSERT outside of a transaction. On the other hand, save() method is not good in a long-running conversation with an extended Session context

Question 178: What is Session and SessionFactory in Hibernate?

Answer: SessionFactory creates and manages the Session objects.

Some points about SessionFactory:

- it is one instance per datasource/database
- it is thread-safe
- it is an immutable and heavy-weight object as it maintains Sessions, mappings, hibernate configurations etc.
- SessionFactory provides second level cache in hibernate also called application-level cache

Some points about Session:

- Session objects are created using sessionFactory.openSession()
- It is one instance per client/thread/transaction
- It is not thread-safe
- It is light-weight
- Session provides first level cache, which is short-lived

Question 179: What is First Level and Second Level Cache in Hibernate?

Answer: Hibernate framework provides caching at two levels, first-level cache which is at the Session level and second-level cache which is at the application level.

The **first level cache** minimizes the database access for the same object if it is requested from the same Session. The first level cache is by default enabled. When you call session.get() method then it hits the database, and while returning, it also saves this object in the first-level cache. So, the subsequent requests for this same object from the same session will not hit the database and the object from cache will be used.

But, since this cache is associated with the Session object, which is a short-lived object in Hibernate, as soon as the session is closed, all the information held in the cache is also lost. So, if we try to load the same object using the get() method, Hibernate will go to the database again and fetch the record.

This poses a significant performance challenge in an application where multiple sessions are used, Hibernate provides second-level cache for this and it can be shared among multiple sessions.

The **second level cache** is maintained at the SessionFactory level, this cache is by default disabled, to enable second level cache in hibernate, it needs to be configured in hibernate configuration file, i.e. *hibernate.cfg.xml* file. There are various providers of second level cache, like EhCache, OSCache etc.

Once second level cache is configured, then object request will first go to the first-level cache, if it is not found there, then it will look for this object in second-level cache, if found then it will be returned from the second-level cache and it will also save a copy in first-level cache.

But, If the object is not found in the second-level cache also, then it will hit the database and if it present in database, this object will be put into both first and second level cache, so that if any other session requests for this object then it will be returned from the cache.

Question 180: What is session.flush() method in Hibernate?

Answer: Flushing the session forces Hibernate to synchronize the in-memory state of the Session with the database. By default, Hibernate will flush changes automatically for you:

- before some query executions
- when a transaction is committed
- when session.flush is called explicitly

It is also possible to define a flushing strategy, by using *FlushMode* class. It provides below modes:

- ALWAYS: the session is flushed before every query
- AUTO: the Session is sometimes flushed before query execution in order to ensure that queries never return stale state
- COMMIT: the Session is flushed when Transaction.commit() is called
- MANUAL: the Session is only ever flushed when Session.flush() is explicitly called by the application

Question 181: How can we see the SQL query that gets generated by Hibernate?

Answer: If you are using *hibernate.cfg.xml* file, then use the below property:

```
<property name="show_sql">true</property>
```

If you are using Spring Data JPA, then you can set this property in application.properties file, like:

```
spring.jpa.show-sql=true
```

Question 182: What is Hibernate Dialect and why we need to configure it?

Answer: The Dialect specifies the type of database that our application is using. As we know, Hibernate is database agnostic and it can work with many databases. However, each database has some variations and standard implementations. We need to tell Hibernate about it, so that Hibernate can generate the database specific SQL wherever it is necessary.

You can configure this dialect in hibernate.cfg.xml file, like:

```
<property name="dialect">org.hibernate.dialect.PostgreSQLDialect</property>
```

And in SpringBoot, you can configure it in application.properties file, like:

```
spring.jpa.database-platform=org.hibernate.dialect.MySQL5InnoDBDialect
```

Question 183: What do you know about hibernate.hbm2ddl.auto property in Hibernate?

Answer: *hibernate.hbm2ddl.auto* automatically validates or exports schema DDL to the database when the SessionFactory is created.

This property takes various values:

- create: creates the schema, but previous data will be lost
- validate: validates the schema
- update: updates the schema. This does not drop any existing table, so we don't lose any existing data. If any column is added in hibernate entity, it will add this column in database table or if any new entity has been added, it will create a new table in database
- create-drop: when this value is given, then the schema is first created at SessionFactory startup and gets dropped at SessionFactory shutdown
- none: it does nothing with the Schema, makes no changes to the database

You can configure this property in hibernate.cfg.xml file, like:

```
<property name="hbm2ddl.auto">create</property>
```

And in SpringBoot, you can configure it in application.properties file, like:

```
spring.jpa.hibernate.ddl-auto=update
```

Maven

Question 184: What is Maven?

Answer: Maven is a tool that is used for building and managing any Java based project. It is a powerful project management tool that is based on POM (Project Object Model). It simplifies the build process.

Question 185: What is pom.xml?

Answer: POM stands for Project Object Model, it is an xml file which contains the configuration information related to the project. Maven uses this file to build the project. We specify all the dependencies that are needed for a project, the plugins, goals etc. By using <packaging> tag, we can specify whether we need to build the project into a JAR/WAR etc.

Question 186: What is local repo and central repo?

Answer: **Local repo**: Local repository is a directory on the developer machine. This repository contains all the dependencies that are downloaded by Maven. The dependencies are downloaded only once even if it is used in multiple projects. By default, local repository location is, C:/Users/USER_NAME/.m2

Central Repo: if any dependency that is required by a project is not found in local repository then Maven looks in central repository for this dependency, then Maven downloads this dependency into the local repository.

We also have one **Remote repository**, which resides on a server from which Maven can download the dependencies into the local repository. It is mainly used in organizations, to share the dependencies within the organization teams.

Question 187: Where we define our local repo path?

Answer: settings.xml file is used to define a local repository location. This file is also used to define proxies, remote repository server locations, plugin groups, profiles etc. By default, it is present in ~/.m2/settings.xml

```
<!-- localRepository
 | The path to the local repository maven will use to store artifacts.
 |
 | Default: ~/.m2/repository -->
<localRepository>/path/to/local/repo</localRepository>
```

Question 188: Where do we define proxies so that maven can download jars from the internet in a corporate environment?

Answer: settings.xml file is used to define proxies which helps in connecting to a network while working in a corporate environment.

```
<proxies>
  <!-- proxy
   | Specification for one proxy, to be used in connecting to the network.-->
  <proxy>
    <id>optional</id>
    <active>true</active>
    <protocol>http</protocol>
    <username>proxyuser</username>
    <password>proxypass</password>
    <host>proxy.host.net</host>
    <port>80</port>
    <nonProxyHosts>local.net|some.host.com</nonProxyHosts>
  </proxy>

</proxies>
```

Question 189: Explain Maven build life-cycle

Answer: Maven build life-cycle is made up of below phases,

- validate: validate the project is correct and all necessary information is available
- compile: compile the source code of the project
- test: test the compiled source code using a suitable unit testing framework. These tests should not require the code to be packaged or deployed
- package: take the compiled code and package it in its distributable format, such as a JAR
- verify: run any checks on results of integration tests to ensure quality criteria's are met
- install: install the package into the local repository, for using as a dependency in other projects locally
- deploy: done in the build environment, copies the final package to the remote repository for sharing with other developers and projects

Maven will first validate the project, then it will try to compile the sources, run the tests against the compiled code, package the binaries (e.g. jar), run integration tests against that package, verify the integration tests, install the verified package to the local repository and then deploy the installed package to a remote repository.

mvn command can be used to execute these build life-cycle phases. If you run *mvn verify,* then it will execute all the phases in order, validate, compile, test, package before calling the *verify*. We only need to call the last build phase.

mvn clean command is used to delete all the project jars that are built by Maven (/target directory of a project). Generally, this clean command is used with install/deploy phase, like *mvn clean deploy* to cleanly build and deploy artifacts into the shared repository.

Database

Question 190: What do you know about SQL Joins?

Answer: SQL joins are used to combine rows from two or more tables, based on a related column between them.

There are 4 types of Joins in SQL:

1. ***Inner join***: Inner join selects all records from Table A and Table B, where the join condition is met.

Syntax:

SELECT Table1.column1, Table1.column2, Table2.column1, ...
FROM Table1
INNER JOIN Table2
On Table1.MatchingColumnName = Table2.MatchColumnName;

(Note: Use either INNER JOIN or JOIN for this operation)

2. ***Left Join***: Left join selects all records from Table A along with records of Table B for which the join condition is met.

Syntax:

SELECT Table1.column1, Table1.column2, Table2.column1, ...
FROM Table1
LEFT JOIN Table2
On Table1.MatchingColumnName = Table2.MatchColumnName;

3. ***Right Join***: Right join selects all records from Table B along with records of Table A for which the join condition is met.

Syntax:

SELECT Table1.column1, Table1.column2, Table2.column1, ...
FROM Table1
RIGHT JOIN Table2
On Table1.MatchingColumnName = Table2.MatchColumnName;

4. ***Full Join***: Full join selects all records from Table A and Table B, regardless of whether the join condition is met or not.

Syntax:

SELECT Table1.column1, Table1.column2, Table2.column1, ...
FROM Table1
FULL JOIN Table2
On Table1.MatchingColumnName = Table2.MatchColumnName;

Question 191: Difference between TRUNCATE & DELETE statements

Answer: The differences are:

- TRUNCATE is a DDL command, whereas DELETE is a DML command
- TRUNCATE removes all rows from a table, whereas DELETE can also remove all rows but it can be used with a 'WHERE' clause to remove specific rows from a table
- TRUNCATE command records very little entry in the transaction log, whereas DELETE statement removes rows one at a time and records an entry in the transaction log for each deleted row, because of this reason TRUNCATE is faster than DELETE
- To use TRUNCATE on a table, you need at least ALTER permission on the table, whereas to use DELETE, you need DELETE permission on the table
- TRUNCATE uses less transaction space than DELETE
- TRUNCATE operation cannot be rolled back, whereas DELETE operation can be rolled back

TRUNCATE command Syntax:

TRUNCATE TABLE employee;

DELETE command Syntax:

DELETE FROM employee; -- delete all rows of the table

DELETE FROM employee WHERE name = 'Mark'; -- delete record where emp_name is Mark

Question 192: Difference between Function and Stored Procedure

Answer: The differences are:

- Function must return a value, whereas Stored Procedure can return zero or n values
- A Function can have only input parameters, whereas a Stored Procedure can have both input and output parameters
- Functions can be called from a Stored Procedure, but a Stored Procedure cannot be called from a Function
- In a Function, DML statements cannot be used, whereas DML statements can be used in a Stored Procedure
- Functions does not allow the usage of try-catch blocks, whereas in Stored Procedure, try-catch block can be used for exception handling
- Transactions are not allowed within Functions, whereas transactions can be used within Stored Procedure
- In a Function, we can only use the table variables, it does not allow using the temporary tables, whereas in a Stored Procedure, both table variables and temporary tables can be used
- Functions can be used in SELECT statement, WHERE clause, HAVING clause, whereas Stored Procedure cannot be used with these
- A Function can be used in JOIN clause as a result set, whereas a Stored Procedure cannot be used in JOIN clause

Question 193: What is DDL, DML statements?

Answer: DDL: DDL stands for Data Definition Language. These statements are used to define the database structure or schema. DDL commands are auto-committed.

Examples of DDL commands are: CREATE, ALTER, DROP, TRUNCATE

DML: DML stands for Data Manipulation language. These statements allows us to manage the data stored in the database. DML commands are not auto-committed, so they can be rolled back.

Examples of DML commands are: INSERT, UPDATE, DELETE, SELECT

Question 194: How to find the nth highest salary from Employee table

Answer: The query to find nth highest salary is:

```
SELECT name, salary
FROM Employee e1
WHERE N-1 = (SELECT COUNT(DISTINCT salary) FROM Employee e2
WHERE e2.salary > e1.salary);
```

Here, to find the 3rd highest salary, replace N with 3, for 5th highest salary, replace N with 5 and so on.

The DISTINCT keyword is used to deal with the duplicate salaries in the table. The highest salary means no salary is higher than it, the second highest salary means only one salary is higher than it, and similarly Nth highest salary means N-1 salaries are higher than it. This is a generic solution and works in all databases, however it is a little slow because the inner query will run for every row processed by the outer query.

Question 195: Difference between UNION and UNION ALL commands in SQL

Answer: Both UNION and UNION ALL are used to combine results of two separate queries, it could be on a same table or a different table but number of columns should be same in both queries.

The Key difference between them is UNION removes duplicates, whereas UNION ALL keeps the duplicates. Because of this, UNION ALL takes less time, as there is no extra step of removing duplicate rows.

Question 196: Difference between Unique Key and Primary Key in SQL

Answer: Both Unique and Primary keys uniquely identifies each row of a table.

The differences between them are:

- There can be only one primary key in a table, whereas there can be multiple unique keys in the table
- Primary key cannot be null, whereas Unique Keys can be null
- In Primary key, default index is clustered, whereas in Unique key, default index is non-clustered

Question 197: What is the difference between Primary and Foreign key in SQL?

Answer: **Primary key** is used to uniquely identify a row in the table. A table can have only one primary key. Primary key is of two types, simple and composite primary key. A Simple Primary key is made up of just one column, whereas a composite primary key is made up of more than one column.

Primary key also enforces some constraints, like UNIQUE and NOT NULL, which means a table cannot have duplicate primary keys and the key cannot be null.

A **Foreign key** in a table is the primary key of another table. For example, consider 2 tables, Employee & Department. Department table have a primary key dept_id and this primary key can be used as foreign key in Employee table to identify that this employee belongs to this department.

The differences between Primary key and Foreign key are given below:

- Primary key uniquely identify a record in the table, whereas Foreign key is the field in the table that is the primary key of another table
- By default, a clustered index is created on primary key, whereas foreign key do not automatically create an index
- We can have only one primary key in a table, whereas we can have more than one foreign key in a table
- Primary keys does not allow duplicate or Null values, whereas Foreign keys allows both

Question 198: What is the difference between clustered and non-clustered index?

Answer: Indexes are used to speed-up the data retrieval performance. There are 2 types of indexes, clustered and non-clustered index and the difference between them,

- A clustered index defines the order in which data is physically sorted in a table, whereas non-clustered index does not sort the physical data inside the table
- By default, clustered index is automatically created on primary key, whereas non-clustered index can be created on any key
- There can be only one clustered index in a table, whereas there can be any number of non-clustered index in a table
- Data Retrieval is faster using Clustered index than non-clustered index
- Data Update is faster using Non-clustered index than clustered index

- Clustered index does not need any extra space, whereas non-clustered index requires extra space to store the index separately
- The size of clustered index is quite large as compared to non-clustered index

Syntax of creating a custom clustered index:

CREATE CLUSTERED INDEX index_name

ON table_name (column_name ASC);

For example, creating a clustered index on Employee table, where data should be stored in ascending order of age column:

CREATE CLUSTERED INDEX employee_asc_age_index

ON employee(age ASC);

Syntax of creating a custom non-clustered index:

CREATE NONCLUSTERED INDEX index_name

ON table_name (column_name ASC);

For example, creating a non-clustered index on Employee table, where data should be stored in ascending order of name column:

CREATE NONCLUSTERED INDEX employee_asc_name_index

ON employee(name ASC);

Question 199: What is the difference between WHERE and HAVING clause in SQL

Answer: The differences are:

- WHERE clause can be used with SELECT, INSERT, UDPATE and DELETE statements, whereas HAVING clause can only be used with SELECT statement
- WHERE clause is used for filtering the rows and it applies on each and every row, whereas HAVING clause is used to filter groups
- WHERE clause is used before GROUP BY clause, whereas HAVING clause is used after GROUP BY clause. It means that WHERE clause is processed before GROUP BY clause while HAVING clause is executed after groups are created
- Aggregate functions cannot be used in WHERE clause, whereas we can use aggregate functions in HAVING clause

Question 200: How to change the gender column value from Male to Female and Female to Male using single Update statement

Answer: This is also a very common interview question. Mostly, this question is asked in a telephonic round. The question is like, you are given a table, say Employee which has a column named 'Gender' having only "Male" or "Female" strings as values. You have to swap these values like wherever the value is Male, it should become Female, and wherever the value is Female, it should become Male. And you have to do this by writing only one Update statement.

The Update query for this is:

UPDATE Employee SET Gender =

CASE Gender WHEN 'Male' THEN 'Female' WHEN 'Female' THEN 'Male' ELSE Gender END;

Other than these common questions, you can be asked to write a lot of queries which mostly contains Joins, so you should also prepare for those types of database queries.

Question 201: Find first 3 largest numbers in an array

In this question, the interviewer will most probably ask you to not use sorting and pick the first/ last 3 numbers from the array. Instead he will ask you to use one "for loop" to solve this problem.

```
public class FindLargestThree {
    //method to find the largest three numbers from an array
    public static void findLargestThree(int arr[]) {

        if(arr.length < 3) {
            System.out.println("Invalid input, Array size is less than 3");
        }

        int first = Integer.MIN_VALUE;
        int second = Integer.MIN_VALUE;
        int third = Integer.MIN_VALUE;

        for(int i=0; i<arr.length; i++) {
            int current = arr[i];

            if(current > first) {
                third = second;
                second = first;
                first = current;
            } else if(current > second) {
                third = second;
                second = current;
            } else if (current > third) {
                third = current;
            }
        }
        System.out.println("Three largest elements are: " + first
                + ", " + second + ", " + third);
    }

    public static void main(String[] args) {
        int arr[] = {19, 5, 78, 1, 33, 11, 20};
        findLargestThree(arr);
    }
}
```

Output:

```
Three largest elements are: 78, 33, 20
```

Here, the idea is to have 3 numbers and then iterating through the array and finding where the current element of array fits in.

At first, we check whether the current element is greater than first, if true, assign the current element to first number by swapping the values of first, second and third.

When the first condition is not true, then we compare the current element with second largest number to find whether the current number is second largest or not, same goes for third condition.

Question 202: Move all negative numbers at the beginning of an array and all positive numbers at the end

Here, also the interviewer will ask not to use any additional data structure like an extra array and this question can be asked in two ways, whether the sequence of original array elements should be maintained or not, so let's see the program where the sequence is not maintained:

```java
public class RearrangeArrayElements {

    static void rearrange(int arr[]) {
        int j=0, temp;
        for(int i=0; i<arr.length; i++) {
            if(arr[i] < 0) {
                if(i != j) {
                    temp = arr[i];
                    arr[i] = arr[j];
                    arr[j] = temp;
                }
                j++;
            }
        }
    }

    public static void main(String[] args) {
        int arr[] = {-9, 5, 1, -2, -15, 7, 12, -3, 2};
        rearrange(arr);

        for(int i=0; i<arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

Output:

-9 -2 -15 -3 1 7 12 5 2

Here, the idea is to iterate through the array and when a negative number is found, then bring that number to the beginning of the array by swapping it with the first positive number.

As you can see that the output is not maintaining the original sequence of array elements.

Now, let's take a look at the solution which maintains the element sequence:

```java
public class RearrangeArrayElements {

    static void rearrange(int arr[]) {
        int j, current;
        for(int i=0; i<arr.length; i++) {
            current = arr[i];
            //if current element is positive then do nothing
            if(current > 0) {
                continue;
            }
            //if current element is negative, then shift positive
            //numbers of arr[0...i-1], one position to their right
            j = i-1;
            while(j >= 0 && arr[j] > 0) {
                arr[j+1] = arr[j];
                j = j-1;
            }

            arr[j+1] = current;
        }
    }
```

```
    public static void main(String[] args) {
        int arr[] = {-9, 5, 1, -2, -15, 7, 12, -3, 2};
        rearrange(arr);

        for(int i=0; i<arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

Output:

-9 -2 -15 -3 5 1 7 12 2

Now, the output is maintaining the original sequence of array elements.

There are many programmatic, puzzle problems that the interviewer can ask. If you are a beginner, then prepare for programs like Palindrome, Fibonacci, Array problems, String problems, Linked list programs etc. First, try to solve them with whatever brute-force solution that comes to your mind, then try to find its time and space complexity, then try to optimize it. If you are not able to think of a better solution, no problem, look for the optimal solution on the internet.