List of technologies:

- Spring core
- Spring MVC
- Spring boot
- Rest
- Testing
- SQL
- Logging

Spring Core

1. What Is Spring Framework?

Spring is the most broadly used framework for the development of Java Enterprise Edition applications. We can use its extensions for building various web applications, or we may just use its dependency injection in simple standalone applications.

2. What Are the Benefits of Using Spring?

Inversion of Control (IoC): Spring container takes care of wiring dependencies of various objects, instead of creating or looking for dependent objects

Aspect Oriented Programming (AOP): Spring supports AOP to separate business logic from system services

IoC container: it manages Spring Bean life cycle and project specific configurations

MVC framework: that is used to create web applications or RESTful web services, capable of returning XML/JSON responses

Transaction management: reduces the amount of boiler-plate code in JDBC operations, file uploading, etc., either by using Java annotations or by Spring Bean XML configuration file

Exception Handling: Spring provides a convenient API for translating technology-specific exceptions into unchecked exceptions

3. What Spring Sub-Projects Do You Know? Describe Them Briefly.

- **Core** – a key module that provides fundamental parts of the framework, like IoC or DI
- **JDBC** – this module allows you to simplify the work with the database
- **ORM integration** – provides integration layers for popular object-relational mapping APIs, such as JPA, JDO, and Hibernate
- **MVC framework** – a web module implementing the Model View Controller design pattern
- **AOP module** – aspect-oriented programming implementation allows you to separate pass-through logic from business logic

4. What Is Dependency Injection?

Dependency Injection is a general concept stating that you do not create your objects manually but instead describe how they should be created. An IoC container will instantiate required classes if needed.

5. How Can We Inject Beans in Spring?

A few different options exist:

- Setter Injection
- Constructor Injection
- Field Injection

The configuration can be done using XML files or annotations.

6. Which Is the Best Way of Injecting Beans and Why?

The recommended approach is to use constructor arguments for mandatory dependencies and setters for optional ones. Constructor injection allows injecting values to immutable fields and makes testing easier.

7. What Is the Difference between Beanfactory and applicationcontext?

In short, the BeanFactory provides the configuration framework and basic functionality, and the ApplicationContext adds more enterprise-specific functionality

8. What is a Spring Bean

The Spring Beans are Java Objects that are initialized by the Spring IoC container.

9. What Is the Default Bean Scope in Spring Framework?

By default, a Spring Bean is initialized as a singleton.

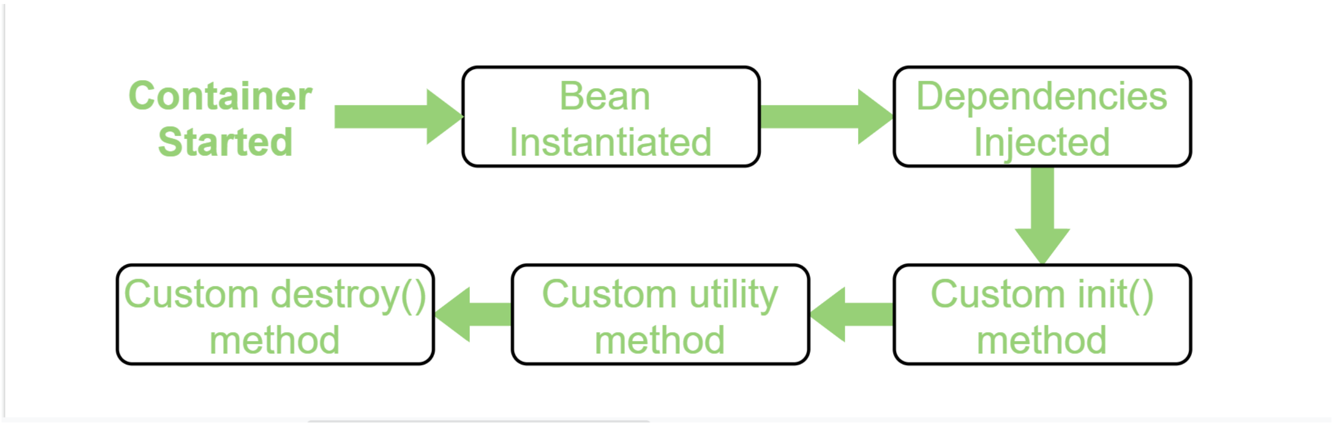
10. How to Define the Scope of a Bean?

To set Spring Bean's scope, we can use @Scope annotation or "scope" attribute in XML configuration files. There are five supported scopes:

- singleton
- prototype
- request
- session
- global-session

12. What Does the Spring Bean Lifecycle Look Like?

Bean life cycle is managed by the spring container. When we run the program then, first of all, the spring container gets started. After that, the container creates the instance of a bean and then dependencies are injected. And finally, the bean is destroyed when the spring container is closed.



13. Can We Have Multiple Spring Configuration Files in One Project?

Yes, in large projects, having multiple Spring configurations is recommended to increase maintainability and modularity.

You can load multiple Java-based configuration files:

```
@Configuration
@Import({MainConfig.class, SchedulerConfig.class})
public class AppConfig {
```

Or load one XML file that will contain all other configs:

```
ApplicationContext context = new ClassPathXmlApplicationContext("spring-all.xml");
```

And inside this XML file you'll have:

```
<import resource="main.xml"/>
<import resource="scheduler.xml"/>
```

14. What Is Spring Boot?

Spring Boot is a project that provides a pre-configured set of frameworks to reduce boilerplate configuration so that you can have a Spring application up and running with the smallest amount of code.

15. Name Some of the Design Patterns Used in the Spring Framework?

- Singleton Pattern: Singleton-scoped beans
- Factory Pattern: Bean Factory classes
- Prototype Pattern: Prototype-scoped beans
- Adapter Pattern: Spring Web and Spring MVC
- Proxy Pattern: Spring Aspect Oriented Programming support
- Template Method Pattern: JdbcTemplate, HibernateTemplate, etc.
- Front Controller: Spring MVC DispatcherServlet
- Data Access Object: Spring DAO support
- Model View Controller: Spring MVC

16. How Does the Scope Prototype Work?

Scope prototype means that every time you call for an instance of the Bean, Spring will create a new instance and return it. This differs from the default singleton scope, where a single object instance is instantiated once per Spring IoC container.

Spring Boot

1. What is Spring Boot and What Are Its Main Features?

Spring Boot is essentially a framework for rapid application development built on Spring Framework. With its auto-configuration and embedded application server support, combined with the extensive documentation and community support it enjoys, Spring Boot is one of the most popular technologies in the Java ecosystem as of date.

Here are a few salient features:

- **Starters** – a set of dependency descriptors to include relevant dependencies at a go

- **Auto-configuration** – a way to automatically configure an application based on the dependencies present on the classpath
- **Embedded server**
- **Security**
- **Logging**

2. What Are the Differences Between Spring and Spring Boot?

The Spring Framework provides multiple features that make the development of web applications easier. These features include dependency injection, data binding, aspect-oriented programming, data access, and many more.

Over the years, Spring has been growing more and more complex, and the number of settings required for such an application can be very large. This is where Spring Boot comes in handy – it makes configuring a Spring application a breeze.

3. How Can We Set up a Spring Boot Application With Maven?

We can include Spring Boot in a Maven project just like we would any other library. However, the best way is to inherit from the spring-boot-starter-parent project and declare dependencies to Spring Boot starters.

Inheriting the spring-boot-starter-parent project is straightforward – we only need to specify a parent element in pom.xml:

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.4.0.RELEASE</version>
</parent>
```

Using the starter parent project is convenient, but not always feasible. For instance, if our company requires all projects to inherit from a standard POM, we can still benefit from Spring Boot's dependency management using a custom parent.

4. What is Spring Initializer?

Spring Initializer is a convenient way to create a Spring Boot project.

We can go to the Spring Initializer site, choose a dependency management tool (either Maven or Gradle), a language (Java, Kotlin or Groovy), a packaging scheme (Jar or War), version and dependencies, and download the project.

5. What Spring Boot Starters Are Available out There?

- Spring-boot-starter: core starter, including auto-configuration support, logging, and YAML
- spring-boot-starter-aop: starter for aspect-oriented programming with Spring AOP and AspectJ
- spring-boot-starter-data-jpa: starter for using Spring Data JPA with Hibernate
- spring-boot-starter-security: starter for using Spring Security
- spring-boot-starter-test: starter for testing Spring Boot applications
- spring-boot-starter-web: starter for building web, including RESTful, applications using Spring MVC

6. How to Deploy Spring Boot Web Applications as Jar?

Spring uses a plugin, namely spring-boot-maven-plugin, to package a web application as an executable JAR. To include this plugin, just add a plugin element to pom.xml:

```
<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
```

With this plugin in place, we'll get a fat JAR after executing the package phase. This JAR contains all the necessary dependencies, including an embedded server. Thus, we no longer need to worry about configuring an external server.

We can then run the application just like we would an ordinary executable JAR.

7. What Are the Basic Annotations that Spring Boot Offers?

The primary annotations that Spring Boot offers in its org.springframework.boot.autoconfigure and its sub-packages. Here are a couple of basic ones:

- @EnableAutoConfiguration – to make Spring Boot look for auto-configuration beans on its classpath and automatically apply them.
- @SpringBootApplication – used to denote the main class of a Boot Application. This annotation combines @Configuration, @EnableAutoConfiguration, and @ComponentScan annotations with their default attributes.

8. Which Embedded Servers does Spring Boot Support, and How to Change the Default?

Spring MVC supports Tomcat, Jetty, and Undertow. Tomcat is the default application server supported by Spring Boot's web starter.

In Spring MVC, to change the default, let's say to Jetty, we need to exclude Tomcat and include Jetty in the dependencies:

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

9. What is thymeleaf?

It is a server side Java template engine for web application. It's main goal is to bring elegant natural templates to your web application.

In order to use Thymeleaf we must add it into our pom.xml file like:

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

It can be integrate with Spring Framework and ideal for HTML5 Java web applications.

10. What is @RestController annotation in Spring Boot?

The @RestController is a stereotype annotation. It adds @Controller and @ResponseBody annotations to the class. We need to import org.springframework.web.bind.annotation package in our file, in order to implement it.

11. What is @RequestMapping annotation in Spring Boot?

The @RequestMappin annotation is used to provide routing information. It tells to the Spring that any HTTP request should map to the corresponding method. We need to import org.springframework.web.annotation package in our file.

12. What do you understand by auto-configuration in Spring Boot and how to disable the auto-configuration?

Auto-configuration is used to automatically configure the required configuration for the application. For example, if you have a data source bean present in the classpath of the application, then it automatically configures the JDBC template. With the help of auto-configuration, you can create a Java application in an easy way, as it automatically configures the required beans, controllers, etc.

To disable the auto-configuration property, you have to exclude attribute of @EnableAutoConfiguration, in the scenario where you do not want it to be applied. @EnableAutoConfiguration(exclude={DataSourceAutoConfiguration.class})

Spring MVC

1. What Is the Role of the @Autowired Annotation?

The @Autowired annotation can be used with fields or methods for injecting a bean by type. This annotation allows Spring to resolve and inject collaborating beans into your bean.

2. Explain a Model Attribute

The *@ModelAttribute* annotation is one of the most important annotations in Spring MVC. It binds a method parameter or a method return value to a named model attribute and then exposes it to a web view.

If we use it at the method level, it indicates the purpose of that method is to add one or more model attributes.

On the other hand, when used as a method argument, it indicates the argument should be retrieved from the model. When not present, we should first instantiate it and then add it to the model. Once present in the model, we should populate the arguments fields from all request parameters that have matching names.

4. Explain the Difference Between @Controller and @RestController?

The main difference between the @Controller and @RestController annotations is that the @ResponseBody annotation is automatically included in the @RestController. This means that we don't need to annotate our handler methods with the @ResponseBody. We need to do this in a @Controller class if we want to write response type directly to the HTTP response body.

5. Describe a PathVariable

We can use the @PathVariable annotation as a handler method parameter in order to extract the value of a URI template variable.

For example, if we want to fetch a user by id from the www.mysite.com/user/123, we should map our method in the controller as /user/{id}:

```
@RequestMapping("/user/{id}")
public String handleRequest(@PathVariable("id") String userId, Model map) {}
```

6. Validation Using Spring MVC

Spring MVC supports JSR-303 specifications by default. We need to add JSR-303 and its implementation dependencies to our Spring MVC application. Hibernate Validator, for example, is one of the JSR-303 implementations.

JSR-303 is a specification of the Java API for bean validation, part of Jakarta EE and JavaSE, which ensures that properties of a bean meet specific criteria, using annotations such as @NotNull, @Min, and @Max.

Spring offers the @Validator annotation and the BindingResult class. The Validator implementation will raise errors in the controller request handler method when we have invalid data. Then we may use the BindingResult class to get those errors.

7. What are the *@RequestBody* and the *@ResponseBody*?

The *@RequestBody* annotation, used as a handler method parameter, binds the HTTP Request body to a transfer or a domain object. Spring automatically deserializes incoming HTTP Request to the Java object using Http Message Converters.

When we use the *@ResponseBody* annotation on a handler method in the Spring MVC controller, it indicates that we'll write the return value of the method directly to the HTTP response body. We'll not put it in a *Model*, and Spring won't interpret as a view name.

8. What is the Purpose of *@EnableWebMVC*?

The *@EnableWebMvc* annotation's purpose is to enable Spring MVC via Java configuration. It's equivalent to *<mvc: annotation-driven>* in an XML configuration. This annotation imports Spring MVC Configuration from *WebMvcConfigurationSupport*. It enables support for *@Controller*-annotated classes that use *@RequestMapping* to map incoming requests to a handler method.

9. What Is ViewResolver in Spring?

The ViewResolver enables an application to render models in the browser – by mapping view names to actual views.

10. What is the *BindingResult*?

BindingResult is an interface from *org.springframework.validation* package that represents binding results. We can use it to detect and report errors in the submitted form. It's easy to invoke — we just need to ensure that we put it as a parameter right after the form object we're validating.

11. What Is the Role of the @Qualifier Annotation?

It is used together with the @Autowired annotation to avoid confusion when multiple instances of a bean type are present.

Let's see an example. We declared two similar beans in XML config:

```
<bean id="person1" class="com.baeldung.Person" >
    <property name="name" value="Joe" />
</bean>
<bean id="person2" class="com.baeldung.Person" >
    <property name="name" value="Doe" />
</bean>
```

When we try to wire the bean, we'll get an NoSuchBeanDefinitionException. To fix it, we need to use @Qualifier to tell Spring about which bean should be wired:

```
@Autowired
@Qualifier("person1")
private Person person;
```

12. What Is the Role of the @Required Annotation?

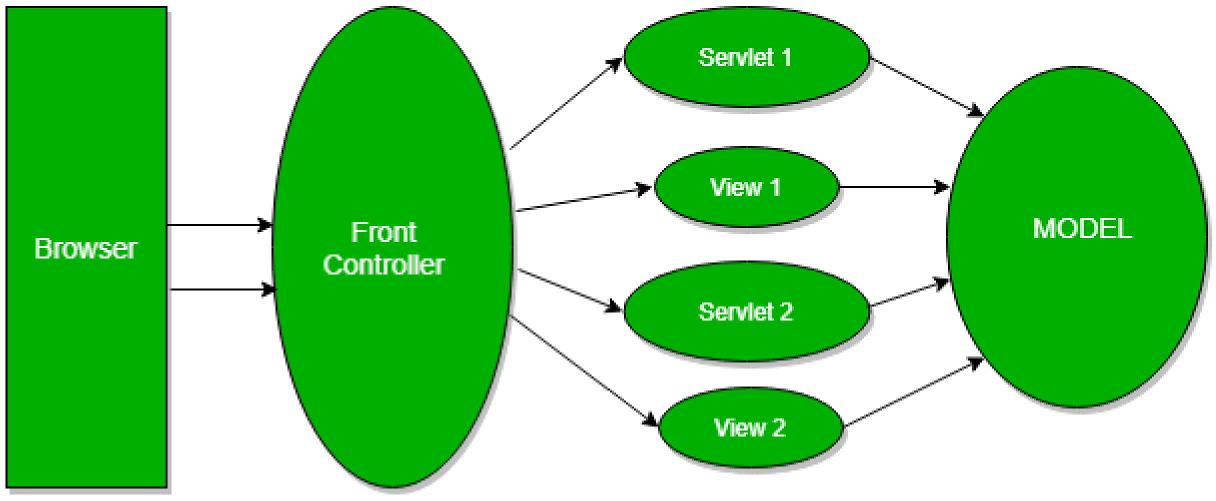
The @Required annotation is used on setter methods, and it indicates that the bean property that has this annotation must be write in at configuration time. Otherwise, the Spring container will throw a BeanInitializationException exception.

Also, @Required differs from @Autowired – as it is limited to a setter, whereas @Autowired is not. @Autowired can be used to wire with a constructor and a field as well, while @Required only checks if the property is set

13.Describe the Front Controller Patter

In the Front Controller pattern, all requests will first go to the front controller instead of the servlet. It'll make sure that the responses are ready and will send them back to the browser. This way we have one place where we control everything that comes from the outside world.

The front controller will identify the servlet that should handle the request first. Then, when it gets the data back from the servlet, it'll decide which view to render and, finally, it'll send the rendered view back as a response:



14. What's the Difference Between @Controller, @Component, @Repository, and @Service Annotations in Spring?

According to the official Spring documentation, @Component is a generic stereotype for any Spring-managed component. @Repository, @Service, and @Controller are specializations of @Component for more specific use cases, for example, in the persistence, service, and presentation layers, respectively.

Let's take a look at specific use cases of last three:

- **@Controller** – indicates that the class serves the role of a controller, and detects @RequestMapping annotations within the class
- **@Service** – indicates that the class holds business logic and calls methods in the repository layer
- **@Repository** – indicates that the class defines a data repository; its job is to catch platform-specific exceptions and re-throw them as one of Spring's unified unchecked exceptions

15. What is DispatcherServlet?

Simply put, in the Front Controller design pattern, a single controller is responsible for directing incoming HttpRequests to all of an application's other controllers and handlers.

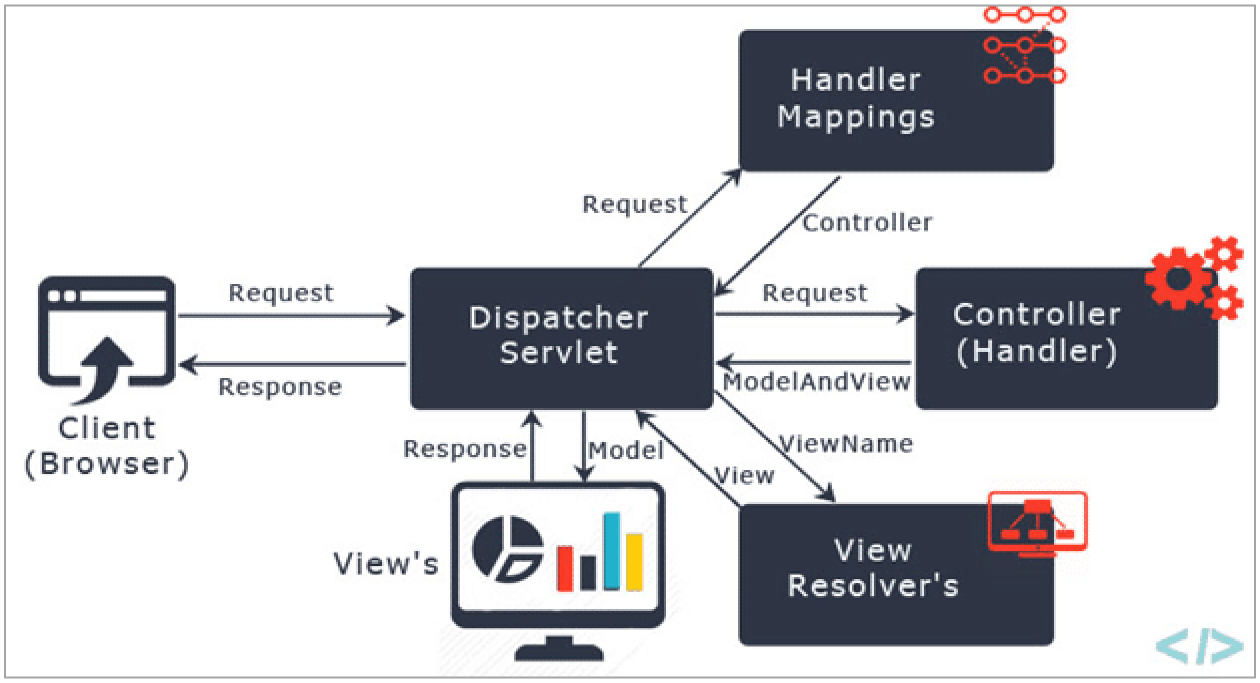
Spring's DispatcherServlet implements this pattern and is, therefore, responsible for correctly coordinating the HttpRequests to the right handlers.

16. What Does the @ExceptionHandler Annotation Do?

The @ExceptionHandler annotation allows us to define a method that will handle the exceptions. We may use the annotation independently, but it's a far better option to use it together with the @ControllerAdvice. Thus, we can set up a global error handling mechanism. In this way, we don't need to write the code for the exception handling within every controller.

17. Explain Spring MVC Architecture.

Spring MVC architecture is based on Model, View, and Controller.



Spring Architecture goes in the following way:

1. The request is received by the dispatcher servlet.
2. Dispatcher servlet sends the request to the handler mapping which provides the response in terms of controller class name.
3. Now the request is sent to the Controller from the dispatcher servlet, hence the controller processes the request and returns the model view object as a response to the dispatcher servlet.
4. Again, the dispatcher servlet sends the request to view resolver to get the correct view page.
5. Lastly, the dispatcher servlet sends the model object received to the browser page to display the result.

Spring Rest

1. What does @RequestMapping annotation do?

The @RequestMapping annotation is used to map web requests to Spring Controller methods. You can map a request based upon HTTP methods, e.g. GET, POST, and various other parameters.

For example, if you are developing a RESTful web service using Spring, then you can use, produce, and consume property along with media type annotations to indicate that this method is only used to produce or consume JSON, as shown below:

```
@RequestMapping
(method = RequestMethod.POST, consumes="application/json")
public Book save(@RequestBody Book aBook) {
  return bookRepository.save(aBook);
}
```

2. When do you need @ResponseBody annotation in Spring MVC?

The @ResponseBody annotation indicates method’s return type should be written directly to the HTTP response body (and not placed in a Model, or interpreted as a view name).

3. What does REST stand for?

REST stands for the REpresentational State Transfer, which uses the HTTP protocol to send data from the client to the server, e.g. a book in the server can be delivered to the client using JSON or XML.

4. What are the advantages of the RestTemplate in Spring MVC?

The RestTemplate class is an implementation of the Template method pattern in the Spring framework. Similar to other popular template classes, like the JdbcTemplate, it also simplifies the interaction with RESTful web services on the client side. You can use it to consume a RESTful web servicer very easily, as shown in this RestTemplate example.

5. What is an HttpMessageConverter in Spring REST?

An HttpMessageConverter is a strategy interface that specifies a converter that can convert from and to HTTP requests and responses. Spring REST uses this interface to convert HTTP responses to various formats, for example, JSON or XML.

Testing

1. What is Unit testing?

Unit testing means testing the smaller units of your application, like classes and methods.
The reason you test your code is to prove to yourself, and perhaps to the users / clients / customers, that your code works.

2. What is JUnit in Java?

Junit is a unit testing framework for the Java programming language. JUnit has been important in the development of test-driven development.

3. What are JUnit main features?

JUnit is an open source framework, which is used for writing and running tests.

- •Provides annotations to identify test methods.
- •Provides assertions for testing expected results.
- •Provides test runners for running tests.
- •JUnit tests allow you to write codes faster, which increases quality.
- •JUnit is elegantly simple. It is less complex and takes less time.

4. What is a Unit Test Case?

A Unit Test Case is a part of code which ensures that the another part of code (method) works as expected.

5.What is the use of @Test annotation?

The @Test annotation is used to mark the method as the test method.

6. What is the difference between manual testing and automated testing?

Manual testing is performed by Human, so it is time-consuming and costly. Automated testing is performed by testing tools or programs, so it is fast and less costly.

7. What is the purpose of org.junit.Assert class?

This class provides a set of assertion methods useful for writing tests. Only failed assertions are recorded.

8. If the JUnit method's return type is 'string', what will happen?

JUnit test methods are designed to return 'void'. So the execution will fail.

9. What is mockito?

Mockito is a JAVA-based library used for unit testing applications. This open-source library plays an important role in automated unit tests for the purpose of test-driven development or behavior-driven development. It uses a mock interface to add dummy functionality in the unit testing. It also uses Java reflection to create mock objects for an interface to test it.

10. List some benefits of Mockito?

Some of the benefits of Mockito are:

- It has support for return values.
- It supports exceptions.
- It has support for the creation of mock using annotation.
- There is no need to write mock objects on your own with Mockito.
- The test code is not broken when renaming interface method names or reordering parameters.

11. What is mocking in testing?

Mocking in Mockito is the way to test the functionality of the class. The mock objects do the mocking process of real service in isolation. These mock objects return the dummy data corresponding to the dummy input passed to the function. The mocking process does not require you to connect to the database or file server to test functionality.

12. What is difference between Assert and Verify in mockito?

Both statements are used to add validations to the test methods in the test suites but they differ in the following.

The Assert command is used to validate critical functionality. If this validation fails, then the execution of that test method is stopped and marked as failed.

In the case of Verify command, the test method continues the execution even after the failure of an assertion statement. The test method will be marked as failed but the execution of remaining statements of the test method is executed normally.

13. List some limitations of Mockito?

Some limitations of the mockito are:

- It cannot mock constructors or static methods.
- It requires Java version 6 plus to run.
- It also cannot mock equals(), hashCode() methods.

14. What is use of mock() method in Mockito?

The Mock() method is used to create and inject the mocked instances. It gives you boilerplate assignments to work with these instances. The other way of creating the instances is using the @mock annotations.

15. What is ArgumentCaptor in Mockito?

ArgumentCaptor is a class that is used to capture the argument values for future assertions. This class is defined in the org.mockito package and can be imported from it.

Some of the methods present in this class are:

•capture(),

- •getValue(),
- •getAllValues(), and ArgumentCaptor <U> forClass

16. What is Hamcrest?

Hamcrest is a framework used for writing customized assertion matchers in the Java programming language. It allows the match rules to be defined declaratively. It can also be used with mock objects by using adaptors. Hamcrest can also used with Junit and TestNG.

17. List some Mockito Annotations?

Some of the Mockito annotations are:

- •@Mock - It is used to create and inject mocked instances.
- •@Spy - It is used to create a real object and spy on the real object.
- •@Captor - It is used to create an ArgumentCaptor.
- •@InjectMocks - It is used to create an object of a class and insert its dependencies.

SQL

1. What is Database?

A database is an organized collection of data, stored and retrieved digitally from a remote or local computer system. Databases can be vast and complex, and such databases are developed using fixed design and modeling approaches.

2. What is SQL?

SQL stands for Structured Query Language. It is the standard language for relational database management systems. It is especially useful in handling organized data comprised of entities (variables) and relations between different entities of the data.

3. What is the difference between SQL and MySQL?

SQL is a standard language for retrieving and manipulating structured databases. On the contrary, MySQL is a relational database management system, like SQL Server, Oracle or IBM DB2, that is used to manage SQL databases.

4. What are Tables and Fields?

A table is an organized collection of data stored in the form of rows and columns. Columns can be categorized as vertical and rows as horizontal. The columns in a table are called fields while the rows can be referred to as records.

5. What are Constraints in SQL?

Constraints are used to specify the rules concerning data in the table. It can be applied for single or multiple fields in an SQL table during creation of table or after creation using the ALTER TABLE command. The constraints are:

- **NOT NULL** - Restricts NULL value from being inserted into a column.
- **CHECK** - Verifies that all values in a field satisfy a condition.
- **DEFAULT** - Automatically assigns a default value if no value has been specified for the field.
- **UNIQUE** - Ensures unique values to be inserted into the field.
- **INDEX** - Indexes a field providing faster retrieval of records.
- **PRIMARY KEY** - Uniquely identifies each record in a table.
- **FOREIGN KEY** - Ensures referential integrity for a record in another table.

6. What is a Primary Key?

The PRIMARY KEY constraint uniquely identifies each row in a table. It must contain UNIQUE values and has an implicit NOT NULL constraint.
A table in SQL is strictly restricted to have one and only one primary key, which is comprised of single or multiple fields (columns).

7. What is a UNIQUE constraint?

A UNIQUE constraint ensures that all values in a column are different. This provides uniqueness for the column(s) and helps identify each row uniquely. Unlike primary key, there can be multiple unique constraints defined per table. The code syntax for UNIQUE is quite similar to that of PRIMARY KEY and can be used interchangeably.

8. What is a Foreign Key?

A FOREIGN KEY comprises of single or collection of fields in a table that essentially refer to the PRIMARY KEY in another table. Foreign key constraint ensures referential integrity in the relation between two tables.
The table with the foreign key constraint is labelled as the child table, and the table containing the candidate key is labelled as the referenced or parent table.

9. What is a Join? List its different types.

The SQL Join clause is used to combine records (rows) from two or more tables in a SQL database based on a related column between the two.

There are four different types of JOINs in SQL:

- (INNER) JOIN: Retrieves records that have matching values in both tables involved in the join. This is the widely used join for queries.
- LEFT (OUTER) JOIN: Retrieves all the records/rows from the left and the matched records/rows from the right table.
- RIGHT (OUTER) JOIN: Retrieves all the records/rows from the right and the matched records/rows from the left table.
- FULL (OUTER) JOIN: Retrieves all the records where there is a match in either the left or right table.

10. What is a Self-Join?

A self JOIN is a case of regular join where a table is joined to itself based on some relation between its own column(s). Self-join uses the INNER JOIN or LEFT JOIN clause and a table alias is used to assign different names to the table within the query.

11. What is an Index? Explain its different types.

A database index is a data structure that provides quick lookup of data in a column or columns of a table. It enhances the speed of operations accessing data from a database table at the cost of additional writes and memory to maintain the index data structure.

There are different types of indexes that can be created for different purposes:

- **Unique and Non-Unique Index**:

Unique indexes are indexes that help maintain data integrity by ensuring that no two rows of data in a table have identical key values. Once a unique index has been defined for a table, uniqueness is enforced whenever keys are added or changed within the index.

Non-unique indexes, on the other hand, are not used to enforce constraints on the tables with which they are associated. Instead, non-unique indexes are used solely to improve query performance by maintaining a sorted order of data values that are used frequently.

- **Clustered and Non-Clustered Index**:
  Clustered indexes are indexes whose order of the rows in the database correspond to the order of the rows in the index. This is why only one clustered index can exist in a given table, whereas, multiple non-clustered indexes can exist in the table.

  The only difference between clustered and non-clustered indexes is that the database manager attempts to keep the data in the database in the same order as the corresponding keys appear in the clustered index.

12. What is a Subquery? What are its types?

A subquery is a query within another query, also known as **nested query** or **inner query**. It is used to restrict or enhance the data to be queried by the main query, thus restricting or enhancing the output of the main query respectively. For example, here we fetch the contact information for students who have enrolled for the maths subject:
There are two types of subquery – **Correlated** and **Non-Correlated**.

- A **correlated** subquery cannot be considered as an independent query, but it can refer the column in a table listed in the FROM of the main query.
- A **non-correlated** subquery can be considered as an independent query and the output of subquery is substituted in the main query.

13. What are UNION, MINUS and INTERSECT commands?

The **UNION** operator combines and returns the result-set retrieved by two or more SELECT statements.
The **MINUS** operator in SQL is used to remove duplicates from the result-set obtained by the second SELECT query from the result-set obtained by the first SELECT query and then return the filtered results from the first.
The **INTERSECT** clause in SQL combines the result-set fetched by the two SELECT

statements where records from one match the other and then returns this intersection of result-sets.
Certain conditions need to be met before executing either of the above statements in SQL -

- Each SELECT statement within the clause must have the same number of columns
- The columns must also have similar data types
- The columns in each SELECT statement should necessarily have the same order

14. What is an Alias in SQL?

It is a temporary name assigned to the table or table column for the purpose of a particular SQL query. In addition, aliasing can be employed as an obfuscation technique to secure the real names of database fields. A table alias is also called a **correlation name**.
An alias is represented explicitly by the **AS** keyword but in some cases the same can be performed without it as well. Nevertheless, using the AS keyword is always a good practice.

Logging

1. Explain what is Log4j?

Log4j is a fast, flexible and reliable logging framework written in Java developed.

2. Explain why to use Apache Log4j?

- Being open-source its completely free to use.
- You can easily save log information into either files or even databases.
- Can be used for projects of any sizes small or large.

3. Mention what are the three principal components of Log4j?

The three principal components of Log4j are

- Loggers

- Appenders
- Layout

4. Inside logger component what are the different log levels?

Different log levels inside logger components are

- All
- Debug
- Info
- Warn
- Error
- Fatal
- Off

5. Explain what are Appenders in Log4j?

Appenders are used to deliver LogEvents to their destination. In simple words, it is used to write the logs in the file.

6. Mention what are the different types of Appenders?

Some of the Appenders type include

- ConsoleAppender logs to standard output
- FileAppender prints logs to some file
- Rolling file appender to a file with maximum size

7. Explain what is layouts in log4j?

Layout in log4j is responsible for formatting logging information in different styles.

8. **Mention what are the two static methods for obtaining a logger object?**

The two static methods for obtaining a logger object are

- Public static Logger getRootLogger()
- Public static Logger getLogger(String name)