1. **Core Java Questions:**

**Below URL for 1-12 Questions**

**https://www.softwaretestinghelp.com/core-java-interview-questions/**

**Q #1)What are the Oops concepts in java?**

**Ans: Oops concepts include:**

* Inheritance
* Encapsulation
* Polymorphism
* Abstraction
* Interface

**Q #2) Difference between HashMap and HashTable.**

**Ans:** **Difference between HashMap and HashTable can be seen below:**

| **HashMap** | **HashTable** |
| --- | --- |
| Methods are not synchronized | Key methods are synchronized |
| Not thread safety | Thread safety |
| Iterator is used to iterate the values | Enumerator is used to iterate the values |
| Allows one null key and multiple null values | Doesn’t allow anything that is null |
| Performance is high than HashTable | Performance is slow |

**Q #3) Difference between Array and Array List.**

**Ans:** **The Difference between Array and Array List can be understood from the below table:**

| **Array** | **Array List** |
| --- | --- |
| Size should be given at the time of array declaration.  String[] name = new String[2] | Size may not be required. It changes the size dynamically.  ArrayList name = new ArrayList |
| To put an object into array we need to specify the index.  name[1] = “book” | No index required.  name.add(“book”) |
| Array is not type parameterized | ArrayList in java 5.0 are parameterized.  Eg: This angle bracket is a type parameter which means a list of String. |

**Q #4) Explain the Priority Queue.**

**Ans: Queue Interface**

**Priority Queue:**Linked list class has been enhanced to implement the queue interface. Queues can be handled with a linked list. Purpose of a queue is “Priority-in, Priority-out”.

Hence elements are ordered either naturally or according to the comparator. The elements ordering represents their relative priority.

**Q #5) What are the types of Exceptions?**

**Ans:** Two types of Exceptions are explained below in detail.

**Checked Exception:**

These exceptions are checked by the compiler at the time of compilation. Classes that extend Throwable class except Runtime exception and Error are called checked Exception.

Checked Exceptions must either declare the exception using throws keyword (or) surrounded by appropriate try/catch.

***E.g.*** ClassNotFound Exception

**Unchecked Exception:**

These exceptions are not checked during the compile time by the compiler.  The compiler doesn’t force to handle these exceptions.

**It includes:**

* Arithmetic Exception
* ArrayIndexOutOfBounds Exception

**Q #6) What are the different ways to handle exceptions?**

**Ans:** **Two different ways to handle exception are explained below:**

**#1) Using try/catch:**

A risky code is surrounded by try block. If an exception occurs, then it is caught by the catch block which is followed by the try block.

**Example:**

|  |  |
| --- | --- |
| 1 | class Manipulation{ |
| 2 | public static void main(String[] args){ | |

|  |  |  |
| --- | --- | --- |
| 3 | add(); | |
| 4 | } |

|  |  |  |
| --- | --- | --- |
| 5 | Public void add(){ | |
| 6 | try{ |

|  |  |
| --- | --- |
| 7 | addition(); |
| 8 | }catch(Exception e){ | |

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | e.printStacktrace(); | | |
| 10 | | } |

|  |  |
| --- | --- |
| 11 | } |
| 12 | } |

**#2) By declaring throws keyword:**

At the end of the method, we can declare the exception using throws keyword.

**Example:**

|  |  |
| --- | --- |
| 1 | class Manipulation{ |
| 2 | public static void main(String[] args){ | |

|  |  |  |
| --- | --- | --- |
| 3 | add(); | |
| 4 | } |

|  |  |  |
| --- | --- | --- |
| 5 | public void add() throws Exception{ | |
| 6 | addition(); |

|  |  |
| --- | --- |
| 7 | } |
| 8 | } |
|  |  |

**Q #7) How do you make a thread in Java?**

**Ans:**There are two ways available in order to make a thread.

**#1) Extend Thread class:**

Extending a Thread class and override the run method. The thread is available in java.lang.thread.

**Example:**

|  |  |  |
| --- | --- | --- |
| 1 | Public class Addition extends Thread { | |
| 2 | public void run () { |

|  |  |
| --- | --- |
| 3 | } |
| 4 | } |

The disadvantage of using a thread class is that we cannot extend any other classes because we have already extend the thread class. We can overload the run () method in our class.

**#2) Implement Runnable interface:**

Another way is implementing the runnable interface. For that we should provide the implementation for run () method which is defined in the interface.

**Example:**

|  |  |  |
| --- | --- | --- |
| 1 | Public class Addition implements Runnable { | |
| 2 | public void run () { |

|  |  |
| --- | --- |
| 3 | } |
| 4 | } |

**Q #8) Explain about join () method.**

**Ans:** Join () method is used to join one thread with the end of the currently running thread.

**Example:**

|  |  |  |
| --- | --- | --- |
| 1 | public static void main (String[] args){ | |
| 2 | Thread t = new Thread (); |

|  |  |  |
| --- | --- | --- |
| 3 | t.start (); | |
| 4 | t.join (); |

|  |  |
| --- | --- |
| 5 | } |

From the above code, the main thread started the execution. When it reaches the code ***t.start()*** then ‘thread t’ starts the own stack for the execution. JVM switches between the main thread and ‘thread t’.

Once it reaches the code ***t.join()*** then ‘thread t’ alone is executed and completes its task, then only main thread started the execution.

It is a non-static method. Join () method has overloaded version. So we can mention the time duration in join () method also “.s”.

**Q #9) Explain thread life cycle in Java.**

**Ans:** **Thread has the following states:**

* New
* Runnable
* Running
* Non-runnable (Blocked)
* Terminated

[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2018/01/Thread-Life-Cycle-in-Java.jpg)

* **New:**

In New state, Thread instance has been created but start () method is not yet invoked. Now the thread is not considered alive.

* **Runnable**:

The Thread is in runnable state after invocation of the start () method, but before the run () method is invoked. But a thread can also return to the runnable state from waiting/sleeping. In this state the thread is considered alive.

* **Running**:

The thread is in running state after it calls the run () method. Now the thread begins the execution.

* **Non-Runnable**(Blocked):

The thread is alive but it is not eligible to run. It is not in runnable state but also, it will return to runnable state after some time.

**Example:** wait, sleep, block.

* **Terminated**:

Once the run method is completed then it is terminated. Now the thread is not alive.

**Q #10) What is Synchronization?**

**Ans:** Synchronization makes only one thread to access a block of code at a time. If multiple thread accesses the block of code, then there is a chance for inaccurate results at the end. To avoid this issue, we can provide synchronization for the sensitive block of codes.

The synchronized keyword means that a thread needs a key in order to access the synchronized code.

Locks are per objects. Every Java object has a lock. A lock has only one key. A thread can access a synchronized method only if the thread can get the key to the objects lock.

For this, we use “Synchronized” keyword.

**Example:**

public class ExampleThread implements Runnable{

public static void main (String[] args){

Thread t = new Thread ();

t.start ();

}

public void run(){

synchronized(object){

{

}

}

**Q #11) Difference between Serialization and Deserialization in Java.**

**Ans:** **These are the difference between serialization and deserialization in java:**

| **Serialization** | **Deserialization** |
| --- | --- |
| Serialization is the process which is used to convert the objects into byte stream | Deserialization is the opposite process of serialization where we can get the objects back from the byte stream. |
| An object is serialized by writing it an ObjectOutputStream. | An object is deserialized by reading it from an ObjectInputStream. |

**Q #12) What is the purpose of a transient variable?**

**Ans:** Transient variables are not part of the serialization process. During deserialization, the transient variables values are set to default value. It is not used with static variables.

**Example:**

transient int numbers;

Below URL for 13 &14 Questions

https://www.tutorialspoint.com/java/java\_interview\_questions.htm

**Q13)**Why is String class considered immutable?

The String class is immutable, so that once it is created a String object cannot be changed. Since String is immutable it can safely be shared between many threads ,which is considered very important for multithreaded programming.

**Q14)Explain the following line used under Java Program −**

public static void main (String args[ ])

The following shows the explanation individually −

* public − it is the access specifier.
* static − it allows main() to be called without instantiating a particular instance of a class.
* void − it affirns the compiler that no value is returned by main().
* main() − this method is called at the beginning of a Java program.
* String args[ ] − args parameter is an instance array of class String

Below Url for Q15&Q16

https://www.journaldev.com/2366/core-java-interview-questions-and-answers

### 15) What are the important features of Java 8 release?

Java 8 has been released in March 2014, so it’s one of the hot topic in java interview questions. If you answer this question clearly, it will show that you like to keep yourself up-to-date with the latest technologies.

Java 8 has been one of the biggest releases after Java 5 annotations and generics. Some of the important features of Java 8 are:

1. [Interface changes with default and static methods](https://www.journaldev.com/2752/java-8-interface-changes-static-method-default-method)
2. [Functional interfaces and Lambda Expressions](https://www.journaldev.com/2763/java-8-functional-interfaces)
3. [Java Stream API for collection classes](https://www.journaldev.com/2774/java-8-stream)
4. [Java Date Time API](https://www.journaldev.com/2800/java-8-date-localdate-localdatetime-instant)

I strongly recommend to go through above links to get proper understanding of each one of them, also read [Java 8 Features](https://www.journaldev.com/2389/java-8-features-with-examples).

### 16) What is try-with-resources in java?

One of the Java 7 features is the try-with-resources statement for automatic resource management. Before Java 7, there was no auto resource management and we should explicitly close the resource. Usually, it was done in the finally block of a try-catch statement. This approach used to cause memory leaks when we forgot to close the resource.

From Java 7, we can create resources inside try block and use it. Java takes care of closing it as soon as try-catch block gets finished. Read more at [Java Automatic Resource Management](https://www.journaldev.com/592/java-try-with-resources).

JDBC Questions:

17) What is JDBC Driver?

JDBC Driver is a software component that enables Java application to interact with the database. There are 4 types of JDBC drivers:

1. **JDBC-ODBC bridge driver:** The JDBC-ODBC bridge driver uses the ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of the thin driver. It is easy to use and can be easily connected to any database.
2. **Native-API driver (partially java driver):** The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in Java. Its performance is better than JDBC-ODBC bridge driver. However, the native driver must be installed on each client machine.
3. **Network Protocol driver (fully java driver):** The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is entirely written in Java. There is no requirement of the client-side library because of the application server that can perform many tasks like auditing, load balancing, logging, etc.
4. **Thin driver (fully java driver):** The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as the thin driver. It is entirely written in Java language. Its performance is better than all other drivers however these drivers depend upon the database.

18) What are the steps to connect to the database in java?

The following steps are used in database connectivity.

* **Registering the driver class:**

The forName() method of the Class class is used to register the driver class. This method is used to load the driver class dynamically. Consider the following example to register OracleDriver class.

* 1. Class.forName("oracle.jdbc.driver.OracleDriver");
* **Creating connection:**

The getConnection() method of DriverManager class is used to establish the connection with the database. The syntax of the getConnection() method is given below.

* 1. 1) **public** **static** Connection getConnection(String url)**throws** SQLException
  2. 2) **public** **static** Connection getConnection(String url,String name,String password)
  3. **throws** SQLException

Consider the following example to establish the connection with the Oracle database.

* 1. Connection con=DriverManager.getConnection(
  2. "jdbc:oracle:thin:@localhost:1521:xe","system","password");
* **Creating the statement:**

The createStatement() method of Connection interface is used to create the Statement. The object of the Statement is responsible for executing queries with the database.

* 1. **public** Statement createStatement()**throws** SQLException

consider the following example to create the statement object

* 1. Statement stmt=con.createStatement();
* **Executing the queries:**

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

Syntax of executeQuery() method is given below.

* 1. **public** ResultSet executeQuery(String sql)**throws** SQLException

Example to execute the query

* 1. ResultSet rs=stmt.executeQuery("select \* from emp");
  2. **while**(rs.next()){
  3. System.out.println(rs.getInt(1)+" "+rs.getString(2));
  4. }

However, to perform the insert and update operations in the database, executeUpdate() method is used which returns the boolean value to indicate the successful completion of the operation.

* **Closing connection:**

By closing connection, object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

Syntax of close() method is given below.

* 1. **public** **void** close()**throws** SQLException

Consider the following example to close the connection.

* 1. con.close();

### 19)How to insert an image or raw data into database?

We can use BLOB to insert image or raw binary data into database.

Spring Interview Questions:

Below URL for Q20-Q25

https://www.journaldev.com/2696/spring-interview-questions-and-answers

**20) What is Spring IoC Container?**

**Inversion of Control** (IoC) is the mechanism to achieve loose-coupling between Objects dependencies. To achieve loose coupling and dynamic binding of the objects at runtime, the objects define their dependencies that are being injected by other assembler objects. Spring IoC container is the program that injects dependencies into an object and makes it ready for our use.

Spring Framework IoC container classes are part of org.springframework.beans and org.springframework.contextpackages and provides us different ways to decouple the object dependencies.

Some of the useful ApplicationContext implementations that we use are;

* AnnotationConfigApplicationContext: For standalone java applications using annotations based configuration.
* ClassPathXmlApplicationContext: For standalone java applications using XML based configuration.
* FileSystemXmlApplicationContext: Similar to ClassPathXmlApplicationContext except that the xml configuration file can be loaded from anywhere in the file system.
* AnnotationConfigWebApplicationContext and XmlWebApplicationContext for web applications.

### 21) What are different scopes of Spring Bean?

There are five scopes defined for Spring Beans.

1. [**singleton**](https://www.journaldev.com/1377/java-singleton-design-pattern-best-practices-examples): Only one instance of the bean will be created for each container. This is the default scope for the spring beans. While using this scope, make sure spring bean doesn’t have shared instance variables otherwise it might lead to data inconsistency issues because it’s not thread-safe.
2. **prototype**: A new instance will be created every time the bean is requested.
3. **request**: This is same as prototype scope, however it’s meant to be used for web applications. A new instance of the bean will be created for each HTTP request.
4. **session**: A new bean will be created for each HTTP session by the container.
5. **global-session**: This is used to create global session beans for Portlet applications.

Spring Framework is extendable and we can create our own scopes too, however most of the times we are good with the scopes provided by the framework.

To set spring bean scopes we can use “scope” attribute in bean element or @Scope annotation for annotation based configurations.

### 22) What’s the difference between @Component, @Controller, @Repository & @Service annotations in Spring?

**@Component** is used to indicate that a class is a component. These classes are used for auto-detection and configured as bean when annotation based configurations are used.

**@Controller** is a specific type of component, used in MVC applications and mostly used with RequestMapping annotation.

**@Repository** annotation is used to indicate that a component is used as repository and a mechanism to store/retrieve/search data. We can apply this annotation with DAO pattern implementation classes.

**@Service** is used to indicate that a class is a Service. Usually, the business facade classes that provide some services are annotated with this.

We can use any of the above annotations for a class for auto-detection but different types are provided so that you can easily distinguish the purpose of the annotated classes.

### 23) How to handle exceptions in Spring MVC Framework?

Spring MVC Framework provides the following ways to help us achieving robust exception handling.

1. **Controller Based** – We can define exception handler methods in our controller classes. All we need is to annotate these methods with @ExceptionHandler annotation.
2. **Global Exception Handler** – Exception Handling is a cross-cutting concern and Spring provides @ControllerAdvice annotation that we can use with any class to define our global exception handler.
3. **HandlerExceptionResolver implementation** – For generic exceptions, most of the times we serve static pages. Spring Framework provides HandlerExceptionResolver interface that we can implement to create global exception handler. The reason behind this additional way to define global exception handler is that Spring framework also provides default implementation classes that we can define in our spring bean configuration file to get spring framework exception handling benefits.

For a complete example, please read [Spring Exception Handling Example](https://www.journaldev.com/2651/spring-mvc-exception-handling-controlleradvice-exceptionhandler-handlerexceptionresolver).

### 24) What are some of the important Spring annotations you have used?

Some of the Spring annotations that I have used in my project are:

* **@Controller** – for controller classes in Spring MVC project.
* **@RequestMapping** – for configuring URI mapping in controller handler methods. This is a very important annotation, so you should go through [Spring MVC RequestMapping Annotation Examples](https://www.journaldev.com/3358/spring-requestmapping-requestparam-pathvariable-example)
* **@ResponseBody** – for sending Object as response, usually for sending XML or JSON data as response.
* **@PathVariable** – for mapping dynamic values from the URI to handler method arguments.
* **@Autowired** – for autowiring dependencies in spring beans.
* **@Qualifier** – with @Autowired annotation to avoid confusion when multiple instances of bean type is present.
* **@Service** – for service classes.
* **@Scope** – for configuring scope of the spring bean.
* **@Configuration**, **@ComponentScan** and **@Bean** – for java based configurations.
* AspectJ annotations for configuring aspects and advices, **@Aspect**, **@Before**, **@After**, **@Around**, **@Pointcut** etc.

### 25) What is Spring Security?

Spring security framework focuses on providing both authentication and authorization in java applications. It also takes care of most of the common security vulnerabilities such as CSRF attack.

It’s very beneficial and easy to use Spring security in web applications, through the use of annotations such as @EnableWebSecurity. You should go through the following posts to learn how to use the Spring Security framework.

* [Spring Security in Servlet Web Application](https://www.journaldev.com/2715/spring-security-example-tutorial)
* [Spring MVC and Spring Security Integration Example](https://www.journaldev.com/2736/spring-security-example-userdetailsservice)

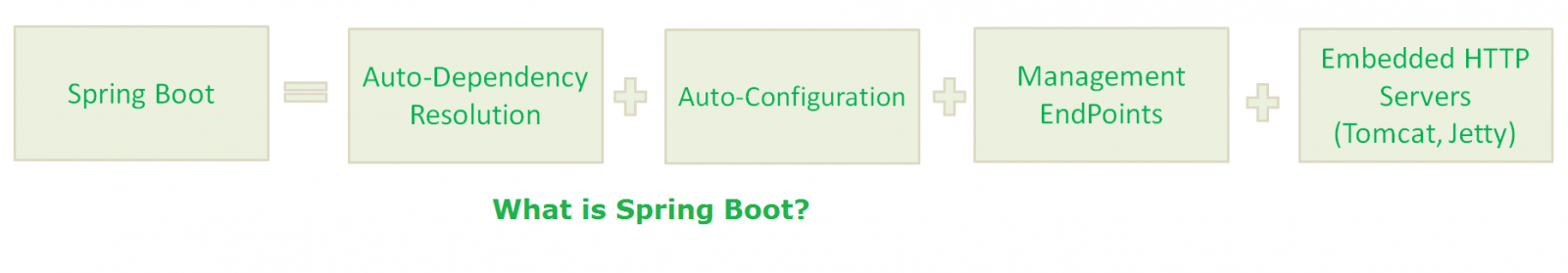
Spring Boot Interview Questions:

Below URL for Q26-Q28

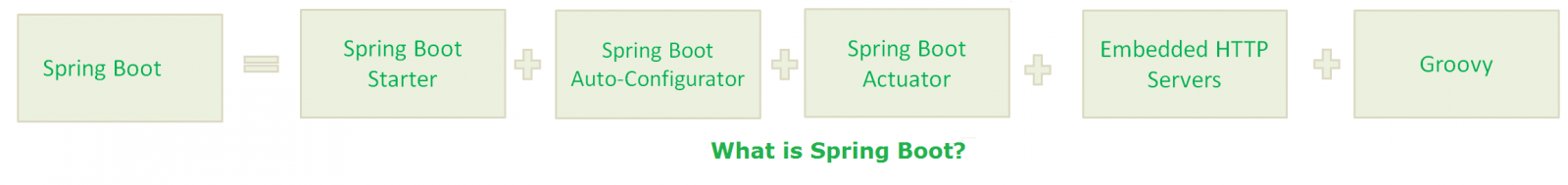
https://www.journaldev.com/8611/spring-boot-interview-questions

### 26)What is Spring Boot?

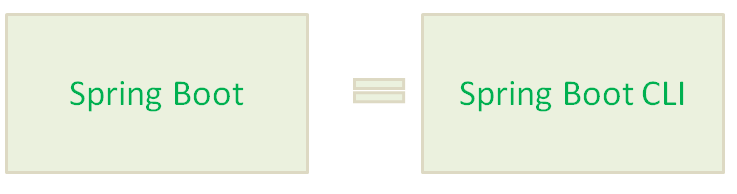
In simple words, Spring Boot Framework is Auto-Dependency Resolution, Auto-Configuration, Management EndPoints, Embedded HTTP Servers(Jetty/Tomcat etc.) and Spring Boot CLI



In other words, Spring Boot Framework is Spring Boot Starter, Spring Boot Auto-Configurator, Spring Boot Actuator, Embedded HTTP Servers, and Groovy.

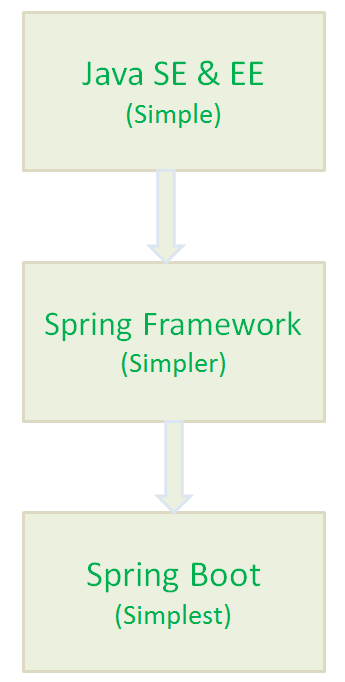


In other words, Spring Boot Framework is Spring Boot CLI.



**NOTE:-**  
If you are confused about the last two definitions, please read the rest of the post to make it clear.

### 27) Why we need Spring Boot?

1. [Spring Framework](https://www.journaldev.com/16922/spring-framework) aims to simplify Java Applications Development.
2. Spring Boot Framework aims to simplify Spring Development.

### 28) Spring Boot Components

Spring Boot Framework has the following components:

1. Spring Boot Starter
2. Spring Boot AutoConfigurator
3. Spring Boot Actuator
4. Spring Boot CLI
5. Spring Boot Initilizr

### ****[29) What is the Spring Boot Initilizr?](https://www.onlineinterviewquestions.com/spring-boot-interview-questions/" \l "collapseUnfiled33)****

**Spring Boot Initilizr** is a web interface which to rapidly create spring boot projects. Using this tool you can create Maven and Gradle projects. You can find Spring Boot Initilizr tool on <https://start.spring.io/>

**30). What are some common Spring Boot annotations?**

Some of the most common Spring Boot annotations are @EnableAutoConfiguration, @SpringBootApplication, @SpringBootConfiguration, and @SpringBootTest.  
  
The @EnableAutoConfiguration is used to enable auto-configuration on Spring Boot application, while @SpringBootApplication is used on the [Main class](http://javarevisited.blogspot.sg/2011/12/main-public-static-java-void-method-why.html) to allow it to run a JAR file. @SpringBootTest is used to run unit test on Spring Boot environment.

**31. Can you control logging with Spring Boot? How?**  
Yes, we can control logging with Spring Boot by specifying log levels on application.properties file. Spring Boot loads this file when it exists in the [classpath](http://www.java67.com/2012/08/what-is-path-and-classpath-in-java-difference.html" \t "_blank) and it can be used to configure both Spring Boot and application code.  
  
Spring Boot uses Commons Logging for all internal logging and you can change log levels by adding following lines in the application.properties file:  
  
logging.level.org.springframework=DEBUG  
logging.level.com.demo=INFO