Java Network programming Interview Questions And Answers Guide.

Global Guideline.

https://globalguideline.com/

Java Network programming Job Interview Preparation Guide.

Question # 1

If you do not want your program to halt while it waits for a connection, put the call to accept( ) in a separate thread?

Answer:-

If you don't want your program to halt while it waits for a connection, put the call to accept( ) in a separate thread.
When you add exception handling, the code becomes somewhat more convoluted. It's important to distinguish between exceptions thrown by the ServerSocket, which should probably shut down the server and log an error message, and exceptions thrown by a Socket, which should just close that active connection. Exceptions thrown by the accept( ) method are an intermediate case that can go either way. To do this, you'll need to nest your try blocks. Finally, most servers will want to make sure that all sockets they accept are closed when they're finished. Even if the protocol specifies that clients are responsible for closing connections, clients do not always strictly adhere to the protocol. The call to close( ) also has to be wrapped in a try block that catches an IOException. However, if you do catch an IOException when closing the socket, ignore it. It just means that the client closed the socket before the server could. Here's a slightly more realistic example:

```
try {
ServerSocket server = new ServerSocket(5776);
      while (true) {
 Socket connection = server.accept(  );
        try {
          OutputStreamWriter out
 = new OutputStreamWriter
    (connection.getOutputStream(  ));
 out.write("You've connected to this server.
      Bye-bye now.rn");
         connection.close(  );
       }
       catch (IOException e) {
 // This tends to be a transitory error for
  this one connection;  e.g. the client
  broke the connection early. Consequently,
// we don't want to break the loop or print
an error message.  However, you might choose
to log this exception in an error log.//
       }
// Most servers will want to guarantee
that sockets are closed
// when complete.
        try {
if (connection != null) connection.close(  );
         }
        catch (IOException e) {}
      }
    }
    catch (IOException e) {
      System.err.println(e);
    }
```

Read More Answers.

Question # 2

Public ServerSocket(int port, int queueLength) throws IOException, BindException?

Answer:-

This constructor creates a ServerSocket on the specified port with a queue length of your choosing. If the machine has multiple network interfaces or IP addresses, then it listens on this port on all those interfaces and IP addresses. The queueLength argument sets the length of the queue for incoming connection requests--that is, how many incoming connections can be stored at one time before the host starts refusing connections. Some operating systems have a maximum queue length, typically five. If you try to expand the queue past that maximum number, the maximum queue length is used instead. If you pass 0 for the port number, the system selects an available port.
For example, to create a server socket on port 5,776 that would hold up to 100 incoming connection requests in the queue, you would write:

```
    try {
ServerSocket httpd = new ServerSocket(5776, 100);
    }
```

```
    catch (IOException e) {
      System.err.println(e);
    }
```

The constructor throws an IOException (specifically, a BindException) if the socket cannot be created and bound to the requested port. An IOException when creating a ServerSocket almost always means one of two things. Either the specified port is already in use, or you do not have root privileges on Unix and you're trying to connect to a port from 1 to 1,023.

Read More Answers.

Question # 3

Explain Look for Local Ports?

Answer:-

Example 11-1: Look for Local Ports

```
   import java.net.*;
   import java.io.*;

 public class LocalPortScanner {

public static void main(String[] args) {

for (int port = 1; port <= 65535; port++)
{

        try {

// the next line will fail and drop into the
catch block if there is already a server
running on the port//
 ServerSocket server = new ServerSocket(port);
        }
        catch (IOException e) {
 System.out.println("There is a server on port
     " + port + ".");
        } // end try

      } // end for

    }

   }
```

Here's the output I got when running LocalPortScanner on my NT workstation:

```
 D:JAVAJNP2examples11>java LocalPortScanner
   There is a server on port 135.
   There is a server on port 1025.
   There is a server on port 1026.
   There is a server on port 1027.
   There is a server on port 1028.
```

Read More Answers.

Question # 4

To create a Socket, you need to know the Internet host to which you want to connect?

Answer:-

To create a Socket, you need to know the Internet host to which you want to connect. When you're writing a server, you don't know in advance who will contact you, and even if you did, you wouldn't know when that host wanted to contact you. In other words, servers are like receptionists who sit by the phone and wait for incoming calls. They don't know who will call or when, only that when the phone rings, they have to pick it up and talk to whoever is there. We can't program that behavior with the Socket class alone. Granted, there's no reason that clients written in Java have to talk to Java servers--in fact, a client doesn't care what language the server was written in or what platform it runs on. However, if Java didn't let us write servers, there would be a glaring hole in its capabilities.

Fortunately, there's no such hole. Java provides a ServerSocket class to allow programmers to write servers. Basically, a server socket's job is to sit by the phone and wait for incoming calls. More technically, a ServerSocket runs on the server and listens for incoming TCP connections. Each ServerSocket listens on a particular port on the server machine. When a client Socket on a remote host attempts to connect to that port, the server wakes up, negotiates the connection between the client and the server, and opens a regular Socket between the two hosts. In other words, server sockets wait for connections while client sockets initiate connections. Once the server socket has set up the connection, the server uses a regular Socket object to send data to the client. Data always travels over the regular socket.

The ServerSocket Class

The ServerSocket class contains everything you need to write servers in Java. It has constructors that create new ServerSocket objects, methods that listen for connections on a specified port, and methods that return a Socket object when a connection is made so that you can send and receive data. In addition, it has methods to set various options and the usual miscellaneous methods such as toString( ).

The basic life cycle of a server is:

1. A new ServerSocket is created on a particular port using a ServerSocket( ) constructor.
2. The ServerSocket listens for incoming connection attempts on that port using its accept( ) method. accept( ) blocks until a client attempts to make a connection, at which point accept( ) returns a Socket object connecting the client and the server.
3. Depending on the type of server, either the Socket's getInputStream( ) method, getOutputStream( ) method, or both are called to get input and output streams that communicate with the client.
4. The server and the client interact according to an agreed-upon protocol until it is time to close the connection.
5. The server, the client, or both close the connection.
6. The server returns to step 2 and waits for the next connection.

Read More Answers.

Question # 5

Explain A Daytime Server?

Answer:-

Example: A Daytime Server

```
   import java.net.*;
   import java.io.*;
   import java.util.Date;

   public class DaytimeServer {

 public final static int DEFAULT_PORT = 13;

 public static void main(String[] args) {

      int port = DEFAULT_PORT;
      if (args.length > 0) {
        try {
       port = Integer.parseInt(args[0]);
   if (port < 0 || port >= 65536) {
System.out.println("Port must between 0 and
    65535");
           return;
         }
       }
       catch (NumberFormatException e) {
         // use default port
       }

      }

      try {

ServerSocket server = new ServerSocket(port);

        Socket connection = null;
        while (true) {

          try {
            connection = server.accept( );
            OutputStreamWriter out
           = new OutputStreamWriter
             (connection.getOutputStream( ));
            Date now = new Date( );
          out.write(now.toString( ) +"rn");
            out.flush( );
            connection.close( );
          }
          catch (IOException e) {}
          finally {
            try {
 if (connection != null) connection.close( );
            }
            catch (IOException e) {}
          }

        }  // end while

      }  // end try
      catch (IOException e) {
        System.err.println(e);
      } // end catch

     } // end main

   } // end DaytimeServer
```

Example is straightforward. The first three lines import the usual packages, java.io and java.net, as well as java.util.Date so we can get the time. There is a single public final static int field (i.e., a constant) in the class DEFAULT_PORT, which is set to the well-known port for a daytime server (port 13). The class has a single method, main( ), which does all the work. If the port is specified on the command-line, then it's read from args[0]. Otherwise, the default port is used.

The outer try block traps any IOExceptions that may arise while the ServerSocket server is constructed on the daytime port or when it accepts connections. The inner try block watches for exceptions thrown while the connections are accepted and processed. The accept( ) method is called within an infinite loop to watch for new connections; like many servers, this program never terminates but continues listening until an exception is thrown or you stop it manually.[1]

When a client makes a connection, accept( ) returns a Socket, which is stored in the local variable connection, and the program continues. We call getOutputStream( ) to get the output stream associated with that Socket and then chain that output stream to a new OutputStreamWriter, out. To get the current date, we construct a new Date object and send it to the client by writing its string representation on out with write( ).

Finally, after the data is sent or an exception has been thrown, we close connection inside the finally block. Always close a socket when you're finished with it. In the previous chapter, we said that a client shouldn't rely on the other side of a connection to close the socket. That goes triple for servers. Clients can time out or crash; users can cancel transactions; networks can go down in high-traffic periods. For any of these or a dozen more reasons, you cannot rely on clients to close sockets, even when the protocol requires them to (which it doesn't in this case).

Read More Answers.

Question # 6

Explain A Time Server?

Answer:-

Sending binary, nontext data is not significantly harder. Example demonstrates with a time server. This follows the time protocol outlined in RFC 868. When a client connects, the server sends a 4-byte, big-endian, unsigned integer specifying the number of seconds that have passed since 12:00 A.M., January 1, 1900 GMT (the epoch). The current time can be retrieved simply by creating a new Date object. However, since the Date class counts milliseconds since 12:00 A.M., January 1, 1970 GMT rather than seconds since 12:00 A.M., January 1, 1900 GMT, some conversion is necessary.

```
   Example: A Time Server
   import java.net.*;
   import java.io.*;
   import java.util.Date;

   public class TimeServer {

 public final static int DEFAULT_PORT = 37;

 public static void main(String[] args) {

      int port = DEFAULT_PORT;
      if (args.length > 0) {
        try {
      port = Integer.parseInt(args[0]);
    if (port < 0 || port >= 65536) {
 System.out.println
      ("Port must between 0 and 65535");
           return;
         }
       }
       catch (NumberFormatException e) {}
      }

// The time protocol sets the epoch at 1900,
 // the java Date class at 1970. This number
// converts between them.

long differenceBetweenEpochs = 2208988800L;

 try {
 ServerSocket server = new ServerSocket(port);
         while (true) {
           Socket connection = null;
           try {
OutputStream out = connection.getOutputStream( );
             Date now = new Date( );
         long msSince1970 = now.getTime( );
         long secondsSince1970 = msSince1970/1000;
         long secondsSince1900 = secondsSince1970
              + differenceBetweenEpochs;
             byte[] time = new byte[4];
             time[0]
= (byte) ((secondsSince1900 & 0x00000000FF000000L)
 >> 24); time[1]
= (byte) ((secondsSince1900 & 0x0000000000FF0000L)
 >> 16); time[2]
 = (byte) ((secondsSince1900 & 0x000000000000FF00L)
  >> 8);  time[3]
= (byte) (secondsSince1900 & 0x00000000000000FFL);
             out.write(time);
             out.flush( );
           } // end try
           catch (IOException e) {
           } // end catch
           finally {
    if (connection != null) connection.close( );
           }
         }  // end while
       }  // end try
       catch (IOException e) {
         System.err.println(e);
       } // end catch

      } // end main

    } // end TimeServer
```

As with the TimeClient of the previous chapter, most of the effort here goes into working with a data format (32-bit unsigned integers) that Java doesn't natively support.

public void close( ) throws IOException

Read More Answers.

Question # 7

What you know about Random Port?

Answer:-

Example: A Random Port

```
   import java.net.*;
   import java.io.*;

   public class RandomPort {

 public static void main(String[] args) {

       try {
ServerSocket server = new ServerSocket(0);
System.out.println("This server runs on port "
         + server.getLocalPort( ));
       }
       catch (IOException e) {
         System.err.println(e);
       }

     }

   }
```

Here's the output of several runs:

```
D:JAVAJNP2examples11>java RandomPort
This server runs on port 1154
D:JAVAJNP2examples11>java RandomPort
This server runs on port 1155
D:JAVAJNP2examples11>java RandomPort
This server runs on port 1156
```

At least on this VM, the ports aren't really random; but they are at least indeterminate until runtime. Socket Options

The only socket option supported for server sockets is SO_TIMEOUT. SO_TIMEOUT is the amount of time, in milliseconds, that accept( ) waits for an incoming connection before throwing a java.io.InterruptedIOException. If SO_TIMEOUT is 0, then accept( ) will never time out. The default is to never time out.

Using SO_TIMEOUT is rather rare. You might need it if you were implementing a complicated and secure protocol that required multiple connections between the client and the server where some responses needed to occur within a fixed amount of time. Most servers are designed to run for indefinite periods of time and therefore use the default timeout value, which is 0 (never time out). public void setSoTimeout(int timeout) throws SocketException

The setSoTimeout( ) method sets the SO_TIMEOUT field for this server socket object. The countdown starts when accept( ) is invoked. When the timeout expires, accept( ) throws an InterruptedIOException. You should set this option before calling accept( ); you cannot change the timeout value while accept( ) is waiting for a connection. The timeout argument must be greater than or equal to zero; if it isn't, the method throws an IllegalArgumentException. For example:

```
   try {
ServerSocket server = new ServerSocket(2048);
server.setSoTimeout(30000);
// block for no more than 30 seconds
      try {
        Socket s = server.accept( );
        // handle the connection
        // ...
      }
      catch (InterruptedIOException e) {
System.err.println
    ("No connection within 30 seconds");
      }
      finally {
        server.close( );
      }
   catch (IOException e) {
 System.err.println
     ("Unexpected IOException: " + e);
   }
```

public int getSoTimeout( ) throws IOException

The getSoTimeout( ) method returns this server socket's current SO_TIMEOUT value. For example:

```
public void printSoTimeout(ServerSocket server)
{

      int timeout = server.getSoTimeOut( );
      if (timeout > 0) {
System.out.println(server + " will time out after "
        + timeout + "milliseconds.");
      }
System.out.println(server + " will never time out.");
      }
      else {
System.out.println("Impossible condition occurred in "
     + server);
System.out.println("Timeout cannot be less than zero." );
      }

   }
```

Read More Answers.

Question # 8

What is a Client Tester?

Answer:-

Example is a program called ClientTester that runs on a port specified on thecommand-line, shows all data sent by the client, and allows you to send a response to the client by typing it on the command line. For example, you can use this program to see the commands that Netscape Navigator sends to a server.
NOTE: Clients are rarely as forgiving about unexpected server responses as servers are about unexpected client responses. If at all possible, try to run the clients that connect to this program on a Unix system or some other platform that is moderately crash-proof. Don't run them on a Mac or Windows 98, which are less stable.
This program uses two threads: one to handle input from the client and the other to send output from the server. Using two threads allows the program to handle input and output simultaneously: it can be sending a response to the client while receiving a request--or, more to the point, it can send data to the client while waiting for the client to respond. This is convenient because different clients and servers talk in unpredictable ways. With some protocols, the server talks first; with others, the client talks first. Sometimes the server sends a one-line response; often, the response is much larger. Sometimes the client and the server talk at each other simultaneously. Other times, one side of the connection waits for the other to finish before it responds. The program must be flexible enough to handle all these cases.
Example shows the code.

```
   Example : A Client Tester
   import java.net.*;
   import java.io.*;
import com.macfaq.io.SafeBufferedReader;
    //


public class ClientTester {

 public static void main(String[] args) {

       int port;

       try {
         port = Integer.parseInt(args[0]);
       }
       catch (Exception e) {
         port = 0;
       }

       try {
 ServerSocket server = new ServerSocket
     (port, 1);
System.out.println
  ("Listening for connections on port "
        + server.getLocalPort( ));

        while (true) {
 Socket connection = server.accept( );
          try {
System.out.println
("Connection established with "+ connection);
Thread input = new InputThread
   (connection.getInputStream( ));
            input.start( );
            Thread output
 = new OutputThread
   (connection.getOutputStream( ));
            output.start( );
// wait for output and input to finish
            try {
              input.join( );
              output.join( );
            }
     catch (InterruptedException e) {
            }
          }
          catch (IOException e) {
            System.err.println(e);
          }
          finally {
            try {
 if (connection !=
    null) connection.close( );
            }
            catch (IOException e) {}
          }
        }
       }
       catch (IOException e) {
         e.printStackTrace( );
       }

     }

   }

class InputThread extends Thread {

     InputStream in;
```

```
 public InputThread(InputStream in) {
        this.in = in;
       }

       public void run( ) {

         try {
           while (true) {
             int i = in.read( );
             if (i == -1) break;
             System.out.write(i);
           }
         }
         catch (SocketException e) {
  // output thread closed the socket
         }
         catch (IOException e) {
           System.err.println(e);
         }
         try {
           in.close( );
         }
         catch (IOException e) {
         }

       }

     }

 lass OutputThread extends Thread {

      Writer out;

public OutputThread(OutputStream out) {
   this.out = new OutputStreamWriter(out);
      }

      public void run( ) {

        String line;
        BufferedReader in
= new SafeBufferedReader
(new InputStreamReader(System.in));
        try {
          while (true) {
            line = in.readLine( );
       if (line.equals(".")) break;
            out.write(line +"rn");
            out.flush( );
          }
        }
        catch (IOException e) {
        }
        try {
          out.close( );
        }
        catch (IOException e) {
        }

       }

     }
```

Read More Answers.

Question # 9

What is an HTTP Server?

Answer:-

HTTP is a large protocol. A full-featured HTTP server must respond to requests for files, convert URLs into filenames on the local system, respond to POST and GET requests, handle requests for files that don't exist, interpret MIME types, launch CGI programs, and much, much more. However, many HTTP servers don't need all of these features. For example, many sites simply display an "under construction" message. Clearly, Apache is overkill for a site like this. Such a site is a candidate for a custom server that does only one thing. Java's network class library makes writing simple servers like this almost trivial.
Custom servers aren't useful only for small sites. High-traffic sites like Yahoo! are also candidates for custom servers because a server that does only one thing can often be much faster than a general purpose server such as Apache or Netscape. It is easy to optimize a special purpose server for a particular task; the result is often much more efficient than a general purpose server that needs to respond to many different kinds of requests. For instance, icons and images that are used repeatedly across many pages or on high-traffic pages might be better handled by a server that read all the image files into memory on startup, and then served them straight out of RAM rather than having to read them off disk for each request. Furthermore, this server could avoid wasting time on logging if you didn't want to track the image request separately from the requests for the pages they were included in.
Finally, Java isn't a bad language for feature-full web servers meant to compete with the likes of Apache or AOLServer. Although CPU-intensive Java programs are demonstrably slower than CPU-intensive C and C++ programs, even when run under a JIT, most HTTP servers are limited by bandwidth, not by CPU speed.

Consequently, Java's other advantages, such as its half-compiled/half-interpreted nature, dynamic class loading, garbage collection, and memory protection, really get a chance to shine. In particular, sites that make heavy use of dynamic content through CGI scripts, PHP pages, or other mechanisms can often run much faster when reimplemented on top of a pure or mostly pure Java web server. Indeed, there are several production web servers written in Java such as the W3C's testbed server Jigsaw (http://www.w3.org/Jigsaw/). Many other web servers written in C now include substantial Java components to support the Java Servlet API and Java Server Pages. On many sites, these are replacing the traditional CGIs, ASPs, and server-side includes, mostly because the Java equivalents are faster and less resource-intensive.
Investigation of HTTP servers begins with a server that always sends out the same file, no matter who or what the request. This is shown in Example, SingleFileHTTPServer. The filename, local port, and content encoding are read from the command line. If the port is omitted, port 80 is assumed. If the encoding is omitted, ASCII is assumed.

Read More Answers.

Question # 10

What is an HTTP Redirector?

Answer:-

Another simple but useful application for a special-purpose HTTP server is redirection. In this section, we develop a server that redirects users from one web site to another--for example, from cnet.com to home.cnet.com. Example reads a URL and a port number from the command-line, opens a server socket on the port, then redirects all requests that it receives to the site indicated by the new URL, using a 302 FOUND code. Chances are this server is fast enough not to require multiple threads. Nonetheless, threads might be mildly advantageous, especially on a high-volume site on a slow network connection. And this server does a lot of string processing, one of Java's most notorious performance bottlenecks.

Read More Answers.

Question # 11

What is JHTTP Web Server?

Answer:-

Example shows the main JHTTP class. As in the previous two examples, the main( ) method of JHTTP handles initialization, but other programs could use this class themselves to run basic web servers.
Example: The JHTTP Web Server

```
import java.net.*;
  import java.io.*;
  import java.util.*;

public class JHTTP extends Thread {

private File documentRootDirectory;
private String indexFileName = "index.html";
private ServerSocket server;
private int numThreads = 50;

public JHTTP(File documentRootDirectory
, int port,
String indexFileName) throws IOException
{

if (!documentRootDirectory.isDirectory( ))
{
throw new IOException(documentRootDirectory
 + " does not exist as a directory");
      }
this.documentRootDirectory =
    documentRootDirectory;
this.indexFileName = indexFileName;
this.server = new ServerSocket(port);
    }

public JHTTP(File documentRootDirectory, int port)
      throws IOException {
this(documentRootDirectory, port, "index.html");
    }

public JHTTP(File documentRootDirectory)
throws IOException {
this(documentRootDirectory, 80, "index.html");
    }

    public void run( ) {

for (int i = 0; i < numThreads; i++) {
Thread t = new Thread(
new RequestProcessor(documentRootDirectory,
 indexFileName));

        t.start( );
      }
System.out.println
("Accepting connections on port "
       + server.getLocalPort( ));

System.out.println("Document Root:
```

```
 " + documentRootDirectory);
        while (true) {
          try {
Socket request = server.accept(  );
RequestProcessor.processRequest(request);
          }
          catch (IOException e) {
          }
        }

      }

 public static void main(String[] args) {

        // get the Document root
        File docroot;
        try {
          docroot = new File(args[0]);
        }
 catch (ArrayIndexOutOfBoundsException e)
  {
 System.out.println
 ("Usage: java JHTTP docroot port indexfile");
          return;
        }

        // set the port to listen on
        int port;
        try {
port = Integer.parseInt(args[1]);
if (port < 0 || port > 65535) port = 80;
        }
        catch (Exception e) {
          port = 80;
        }

        try {
 JHTTP webserver = new JHTTP(docroot, port);
          webserver.start(  );
        }
        catch (IOException e) {
System.out.println
("Server could not start because of an "
           + e.getClass(  ));
          System.out.println(e);
        }

      }

    }
```

The main( ) method of the JHTTP class sets the document root directory from args[0]. The port is read from args[1], or 80 is used for a default. Then a new JHTTP thread is constructed and started. The JHTTP thread spawns 50 RequestProcessor threads to handle requests, each of which will retrieve incoming connection requests from the RequestProcessor pool as they become available. The JHTTP thread repeatedly accepts incoming connections and puts them in the RequestProcessor pool. .

Read More Answers.

Question # 12

What is a Thread Pool?

Answer:-

Each connection is handled by the run( ) method of the RequestProcessor class shown in Example. This method waits until it can get a Socket out of the pool. Once it does that, it gets input and output streams from the socket and chains them to a reader and a writer. The reader reads the first line of the client request to determine the version of HTTP that the client supports--we want to send a MIME header only if this is HTTP 1.0 or later--and what file is requested. Assuming the method is GET, the file that is requested is converted to a filename on the local filesystem. If the file requested was a directory (i.e., its name ended with a slash), we add the name of an index file. We use the canonical path to make sure that the requested file doesn't come from outside the document root directory. Otherwise, a sneaky client could walk all over the local filesystem by including .. in URLs to walk up the directory hierarchy. This is all we'll need from the client, though a more advanced web server, especially one that logged hits, would read the rest of the MIME header the client sends.
Next the requested file is opened and its contents are read into a byte array. If the HTTP version is 1.0 or later, we write the appropriate MIME headers on the output stream. To figure out the content type, we call the guessContentTypeFromName( ) method to map file extensions such as .html onto MIME types such as text/html. The byte array containing the file's contents is written onto the output stream, and the connection is closed. Exceptions may be thrown at various places if, for example, the file cannot be found or opened. If an exception occurs, we send an appropriate HTTP error message to the client instead of the file's contents

Read More Answers.

Networking Most Popular Interview Topics.

1 : CCNA Frequently Asked Interview Questions and Answers Guide.

2 : MCSE Frequently Asked Interview Questions and Answers Guide.

3 : Active Directory Frequently Asked Interview Questions and Answers Guide.

4 : CCNP Frequently Asked Interview Questions and Answers Guide.

5 : Routing Frequently Asked Interview Questions and Answers Guide.

6 : VPN Frequently Asked Interview Questions and Answers Guide.

7 : Networks and Security Frequently Asked Interview Questions and Answers Guide.

8 : VoIP Frequently Asked Interview Questions and Answers Guide.

9 : CCNA Security Frequently Asked Interview Questions and Answers Guide.

10 : LAN (Local area network) Frequently Asked Interview Questions and Answers Guide.

About Global Guideline.

Global Guideline is a platform to develop your own skills with thousands of job interview questions and web tutorials for fresher's and experienced candidates. These interview questions and web tutorials will help you strengthen your technical skills, prepare for the interviews and quickly revise the concepts. Global Guideline invite you to unlock your potentials with thousands of **Interview Questions with Answers** and much more. Learn the most common technologies at Global Guideline. We will help you to explore the resources of the World Wide Web and develop your own skills from the basics to the advanced. Here you will learn anything quite easily and you will really enjoy while learning. Global Guideline will help you to become a professional and Expert, well prepared for the future.

* This PDF was generated from https://GlobalGuideline.com at **November 29th, 2023**

* If any answer or question is incorrect or inappropriate or you have correct answer or you found any problem in this document then don't hesitate feel free and e-mail us we will fix it.

You can follow us on FaceBook for latest Jobs, Updates and other interviews material.
www.facebook.com/InterviewQuestionsAnswers

Follow us on Twitter for latest Jobs and interview preparation guides
https://twitter.com/InterviewGuide

Best Of Luck.

Global Guideline Team
https://GlobalGuideline.com
Info@globalguideline.com