Branch-CSE

Subject: Advanced Java

|  |  |  |
| --- | --- | --- |
|  | Part–I | |
| 01 | Short Answer Type Questions | |
|  | a) | Write down the different methods of Naming class in RMI.   |  |  | | --- | --- | | public static java.rmi.Remote **lookup**(java.lang.String) | It returns the reference of the remote object. | | public static void **bind**(java.lang.String, java.rmi.Remote) | It binds the remote object with the  given name. | |  |  | | public static void **unbind**(java.lang.String) | It destroys the remote object which is  bound with the given name. | | public static void **rebind**(java.lang.String, java.rmi.Remote) | It binds the remote object to the  new name. | | public static java.lang.String[] **list**(java.lang.String) | It returns an array of the names of the remote objects bound in the registry. | |
|  | b) | What is URL ? What are methods used in URL class?  URL and its Methods  URL: Uniform Resource Locator, a string that identifies a resource on the internet.  Its common methods are **getProtocol(),getHost(),getPort(),getFile()**  URL ur = new URL("http://www.gita.in/academics");  System.out.println("protocol: " +ur.getProtocol()); http  System.out.println("protocol: " +ur.getHost());www.gita.in  System.out.println("protocol: " +ur.getPort()); -1  System.out.println("protocol: " +ur.getFile());/academics |
|  | c) | List out different methods of ResultSet interface.  The important methods of Statement interface are as follows:   |  |  |  | | --- | --- | --- | | **) public boolean next():** | is used to move the cursor to the one row next from the current position. | | | **2) public boolean previous():** | is used to move the cursor to the one row previous from the current  position. | | | **3) public boolean first():** | is used to move the cursor to the first row in result set object. | | | **4) public boolean last():** | is used to move the cursor to the last row in result set object. | | | **public boolean absolute(int row):** | is used to move the cursor to the specified row number in the  ResultSet object. | |  | |  | |  | |
|  | d) | State the differences between Statement and PreparedStatement.   * Statement: Used to execute SQL statements that are not **pre-compiled**. Suitable   for simple, dynamic queries.  String sql="insert into emp values(‘A’,’B’,’c’)";   * PreparedStatement: Used to execute **pre-compiled SQL statements**. More efficient   and secure, especially for repetitive queries.  String sql="insert into emp values(?,?,?)";  **Extra** |

|  |  |
| --- | --- |
| **Statement** | **PreparedStatement** |
| **It is used when SQL query is to be executed only once.** | **It is used when SQL query is to be executed multiple times.** |
| **You can’t pass parameters at runtime.** | **You can pass parameters at runtime.** |
| **Used for CREATE, ALTER, DROP statements.** | **Used for the queries which are to be executed multiple times.** |
| **Performance is very low.** | **Performance is better than Statement.** |
| **It is base interface.** | **It extends statement interface.** |
| **Used to execute normal SQL queries.** | **Used to execute dynamic SQL queries.** |
| **We can’t use statement for reading binary data.** | **We can use Preparedstatement for reading binary data.** |
| **It is used for DDL statements.** | **It is used for any SQL Query.** |
| **We can’ t use statement for writing binary data.** | **We can use Preparedstatement for writing binary data.** |
| **No binary protocol is used for communication.** | **Binary protocol is used for communication.** |

|  |  |  |
| --- | --- | --- |
|  | e) | Which method is overridden in servlet lifecycle ?, justify.  Overridden Servlet Lifecycle methods   * **init(**ServletConfig config): **Called once** when the servlet is first loaded. Used for **initialization tasks**. * **service**(ServletRequest req, ServletResponse resp): Called for each client request. Handles request processing and response generation. * **destroy():** Called once when the servlet is unloaded. Used for **cleanup tasks.** |
|  | f) | WAP in java to get the IP address of any local computer.  **import java.net.InetAddress**;  InetAddress localhost = InetAddress.getLocalHost();  System.out.println("Local IP address: " + localhost.getHostAddress()); |
|  | g) | How can we make a ResultSet scrollable?  • Use ResultSet.TYPE\_SCROLL\_SENSITIVE when **creating the statement.**  Call setFetchDirection(int direction) to specify the direction (e.g., ResultSet.FETCH\_FORWARD or ResultSet.FETCH\_REVERSE).  • Use relative(int rows) to move the **cursor relative** to the current position |
|  | h) | Write the driver’s name used to connect toOracle11g database.  • ojdbc6.jar: Oracle JDBC driver for Oracle 11g. |
|  | i) | Differentiate Statement and CallableStatement. |

|  |  |
| --- | --- |
| **Statement** | **CallableStatement** |
| It is used when SQL query is to be executed only once. | It is used when the stored procedures are to be executed. |
| You can’t pass the parameters at runtime. | You can pass the parameters at runtime. |
| Used for CREATE, ALTER, DROP statements. | Used to execute functions. |
| Performance is very low. | Performance is better than Statement. |
| Used to execute normal SQL queries. | Used to call the stored procedures. |
| It is base interface. | It extends PreparedStatement interface. |
| It is used for DDL statements. | It is used for stored procedures. |
| We can not used statement for reading binary data.. | We can used CallableStatement for reading binary data.. |

|  |  |  |
| --- | --- | --- |
|  | j) | What is DNS? Why it is used?  DNS: Domain Name System  • Translates domain names (e.g., www.google.com) to IP addresses.  • Decentralized system with distributed databases.  • Improves user experience and simplifies internet navigation. |
|  | k) | What is Stored Procedure and how it is different from function?  Stored Procedure vs Function   * Stored Procedure: A set of SQL statements that can be stored and executed as a unit. Can accept input parameters and return output values. * Function: A unit of code that performs a specific task and returns a single value.   **Extra** |

| **Functions** | **Procedures** |
| --- | --- |
| A function has a return type and returns a value. | A procedure does not have a return type. But it returns values using the OUT parameters. |
| You cannot use a function with Data Manipulation queries. Only Select queries are allowed in functions. | You can use [**DML queries**](https://www.tutorialspoint.com/what-are-the-dml-commands-in-dbms) such as [**insert**](https://www.tutorialspoint.com/mysql/mysql_insert.htm), [**update**](https://www.tutorialspoint.com/mysql/mysql_update.htm), [**select**](https://www.tutorialspoint.com/mysql/mysql_select.htm) etc… with procedures. |
| A function does not allow output parameters | A procedure allows both input and output parameters. |
| You cannot manage transactions inside a function. | You can manage transactions inside a procedure. |
| You cannot call stored procedures from a function | You can call a function from a stored procedure. |
| You can call a function using a select statement. | You cannot call a procedure using select statements. |

|  |  |  |
| --- | --- | --- |
|  | l) | Differentiate between TCP/IP and UDP |

|  |  |  |
| --- | --- | --- |
| Feature | TCP/IP | UDP |
| Connection-oriented | Yes | No |
| Reliable delivery | Guaranteed | Not guaranteed |
| Ordered delivery | Yes | No |
| Error checking | Yes | No |
| Flow control | Yes | No |
| Overhead | High | Low |
| Speed | Slower | Faster |
| Applications | File transfer, email, web browsing | Streaming media, online gaming, VoIP |

|  |  |  |
| --- | --- | --- |
|  | m) | Which jdbc driver is used widely in industry and why?  **Type-4 thin driver is used widely because:**  **Performance: They offer the best performance as they directly communicate with the database without any**  **intermediaries, minimizing overhead and latency.**  **Features: Vendor-specific drivers often provide access to advanced features and functionalities not available with**  **generic drivers.**  **Maturity and stability: They are typically well-tested and mature, leading to fewer bugs and more reliable**  **operation.**  **Support: Vendor support is readily available, ensuring easier troubleshooting and problem-solving.** |
|  | n) | The thin driver converts JDBC calls directly into the **vendor-specific database protocol**. That is why  it is known as thin driver. It is fully written in Java language.  Thin driver |
|  | o) | What is static IP and why it is used?  Static IP  A static IP address is a fixed address assigned to a device on a network. This differs from a dynamic IP, which  changes automatically. Static IPs are used for:   * Servers: Always accessible for incoming connections. * Security devices: Need a consistent address for configuration and monitoring. * Printers and other network devices: Easier to locate and manage. |
|  | p) | What is a web container ?  A web container is a software program that manages the **lifecycle of servlets** and provides them with runtime  services. It handles tasks like:  • Loading and unloading servlets  • Creating and initializing servlet instances  • Mapping URLs to servlets  • Handling HTTP requests and responses |
|  | q) | What is HTTP? Why it is used?  HTTP is a protocol for fetching resources such as HTML documents.  8 |
|  | r) | State five server-side technologies. |
|  | s) | What is the name of the servlet container provided by Tomcat?  Catalina is Tomcat's servlet container. Catalina implements Sun Microsystems' specifications for servlet and  JavaServer Pages (JSP).  **Apache Tomcat** is a long-lived, open source Java servlet container that implements core Java enterprise  Specifications. Apache Tomcat is a long-lived, open source Java servlet container that implements core  Java enterprise (now **Jakarta EE**) specifications, |
|  | t) | What is MIME? Write three MIME formats.  MIME stands for **Multipurpose Internet Mail Extensions**. It's a standard for attaching different types of data to  emails, such as images, documents, and audio files.   * text/plain: Plain text document, like a simple **.txt file.** * image/png: Portable Network Graphic, a common image format. * application/pdf: Adobe Portable Document Format, used for documents with text, images, and formatting. |
|  | u) | What is a deployment descriptor?  A **deployment descriptor** is an **XML file** (usually named **web.xml**) used to configure web applications deployed  on a web server like Tomcat. It specifies details like servlets, filters, listeners, context parameters, and  security constraints. |
|  | v) | Differentiate web server and application server?   * web server: Responsible for serving static content like HTML, CSS, and images, and handling basic HTTP requests. Examples: Apache, NGINX, Tomcat (in web server mode). * Application server: Handles dynamic content generation, business logic, and server-side processing.    Examples: Tomcat (in servlet container mode), JBoss, WebLogic. |
|  | w) | Write the advantages of JSP over Servlet. Advantages of JSP over Servlet:   * Easier to learn and use: JSP syntax is closer to HTML, making it more intuitive for web developers. * Separation of concerns: Code and logic are separated from presentation (HTML), leading to cleaner   and more maintainable code.   * Template-based development: JSP promotes reuse of templates and components, enhancing   development speed.   * Expression language: JSP allows embedding expressions directly in the HTML, simplifying   data access and manipulation. |
|  | x) | What is Logging service of a web server?  A logging service records information about web server activities, including requests, errors, security events, and  performance metrics. It helps with troubleshooting, monitoring, and analyzing web application behavior.  Common web server logs include access logs, error logs, and audit logs. |
|  | y) | Write the different types of implicit objects in JSP. Implicit objects are predefined objects available  within JSP pages, providing access to various information and resources:   * request: Represents the current HTTP request. * response: Represents the HTTP response to be sent. * session: Represents the user's session data. * application: Represents the web application context. * out: Stream used to send output to the browser. |
|  | z) | Differentiate between API and Frame work.    **optional**  API (Application Programming Interface):   * Function-oriented: Provides building blocks in the form of functions, classes, and protocols. * Lower-level abstraction: Requires more manual coding and understanding of underlying concepts. * Greater flexibility: Developers have complete control over how they use the API to achieve their goals. * Examples: Google Maps API, Twitter API, Java Servlet API.   Framework:   * Structure-oriented: Offers a pre-written code structure with predefined functionalities and workflows. * Higher-level abstraction: Simplifies development by providing ready-made solutions for common tasks. * Less flexibility: Developers are constrained by the framework's structure and conventions. * Examples: Spring Framework (Java), Django (Python), ReactJS (JavaScript).   **OPtional**  Here's an analogy:   * API: Imagine an API as a toolbox. It provides various tools (functions) you can use to build something,   but you have to figure out how to assemble them yourself.   * Framework: Imagine a framework as a pre-built furniture kit. It provides all the necessary parts and   instructions to assemble a specific piece of furniture, leaving less room for customization but making  it easier and faster to achieve the desired result. |
|  | Part–II | |
| 02 | Focussed–Short answer type Questions | |
|  | a) | What is a socket? How it is used to achieve communication? Explain the steps with suitable program.  A socket is simply **an endpoint for communications between the machines**. The Socket class can be used to  create a socket.  Java **Socket programming** is used for communication between the applications running on  **different JRE**.  Java Socket programming can be **connection-oriented or connection-less**.  **Socket** and **ServerSocket classes** are used for **connection-oriented** socket programming and  **DatagramSocket** and **DatagramPacket** classes are used for connection-less socket programming.  The client in socket programming must know two information:   1. **IP Address of Server**, and 2. **Port number.**   import java.io.\*;  import java.net.\*;  public class MyClient {  public static void main(String[] args) {  try{  Socket s=new Socket("localhost",6666);  DataOutputStream dout=new DataOutputStream(s.getOutputStream());  DataInputStream dis = new DataInputStream(s.getInputStream());  BufferedReader br = new BufferedReader( new InputStreamReader(System.in));  System.out.println("enter ur msg for serverside");  String str = br.readLine();  dout.writeUTF(str);  dout.flush();  String str2 = dis.readUTF();  System.out.println(“server: ”+ str2);  dout.close();  s.close();  }catch(Exception e){System.out.println(e);}  }  }    **2 Way communication serverside**  import java.io.\*;  import java.net.\*;  public class ServerSide2 {      public static void main(String[] args)throws Exception {          ServerSocket ss = new ServerSocket(1234);          Socket s = ss.accept();          DataInputStream dis = new DataInputStream(s.getInputStream());          DataOutputStream dos = new DataOutputStream(s.getOutputStream());          BufferedReader br= new BufferedReader(new InputStreamReader(System.in));          String str = dis.readUTF(); //will store msg sent by client side          System.out.println("client : "+str);          System.out.println("enter ur msg for client");          String str2 = br.readLine();          dos.writeUTF(str2);          dos.flush();          dos.close();          dis.close();          s.close();          ss.close();        }  } |
|  | b) | WAP to delete some rows from EMP table.   1. **import** java.sql.\*; 2. **class** FetchRecord{ 3. **public** **static** **void** main(String args[])**throws** Exception{ 4. Class.forName("oracle.jdbc.driver.OracleDriver"); 5. Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe",   "system","oracle");   1. Statement stmt=con.createStatement(); 2. **int** result=stmt.executeUpdate("delete from emp765 where id=33"); 3. System.out.println(result+" records affected"); 4. con.close(); 5. }} |
|  | c) | Explain marshalling and unmarshalling.  During communication between two machines through RMI, parameters are packed into a  message and then sent over the network. This packing of parameters into a message is called  marshalling. On the other side these packed parameters are unpacked from the message which  is called unmarshalling.  **Stubs** writes and transmits (**marshals)** the parameters to the remote Virtual Machine (JVM),  It reads **(unmarshals)** the return value or exception  **Skeleton** writes and transmits **(marshals)** the result to the caller. |
|  | d) | What is class path? Why do we set the class path temporarily and permanently? Explain with example.  **The classpath is an environment variable used by the Java Virtual Machine**  **(JVM) to locate and load classes when running a Java program**. It specifies a  list of directories,  JAR files, and ZIP files where the JVM should look to find and load class files.   |  | | --- | | There are two ways to set the classpath:   * temporary * permanent |   How to set the temporary classpath:   |  | | --- | | Firstly, search the ojdbc14.jar file then open command prompt and write: |  1. C:>set classpath=c:\folder\ojdbc14.jar;.;   How to set the permanent classpath:  Go to environment variable then click on new tab. In variable name write **classpath** and in  variable value paste the path to ojdbc14.jar by  appending ojdbc14.jar;.; as C:\oraclexe\app\oracle\product\10.2.0\server\jdbc\lib\ojdbc14.jar;.; |
|  | e) | WAP to display the content of EMP table.  **import java.sql.\*;**  **class OracleCon{**  **public static void main(String args[]){**  **try{**  **//step1 load the driver class**  **Class.forName("oracle.jdbc.driver.OracleDriver");**    **//step2 create the connection object**  **Connection con=DriverManager.getConnection(**  **"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");**    **//step3 create the statement object**  **Statement stmt=con.createStatement();**    **//step4 execute query**  **ResultSet rs=stmt.executeQuery("select \* from emp");**  **while(rs.next())**  **System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));**    **//step5 close the connection object**  **con.close();**    **}catch(Exception e){ System.out.println(e);}**    **}**  **}** |

|  |  |  |
| --- | --- | --- |
|  | f) | Why place holders are used in jdbc program, explain with example?  Placeholders, also known  as bind variables, are crucial elements in JDBC programs for several reasons:  1. Security:   * Placeholders help prevent SQL injection attacks. By separating data from the query,    you avoid embedding user input directly into the SQL string. This prevents malicious  users from injecting harmful code through input fields.  Example:  Without placeholders:  SQL  String name = user\_input;  String sql = "SELECT \* FROM users WHERE username = '" + name + "'";  This makes the code vulnerable to SQL injection if user\_input is crafted to include  malicious code.  With placeholders:  SQL  String sql    = "SELECT \* FROM users WHERE username = ?";  PreparedStatement stmt = connection.prepareStatement(sql);  stmt.setString(1, user\_input);  Here, the user input is set as a parameter separate from the query. The database engine  handles the data and prevents injection attempts.  2. Reusability:   * Placeholders enable reusing the same prepared statement with different data sets.   This reduces redundant code and improves performance.  Example:  You need to fetch user data for multiple usernames. Instead of creating separate statements  for each username, you can use a single prepared statement with the placeholder:  SQL  String sql = "SELECT \* FROM users WHERE username = ?";  PreparedStatement stmt = connection.prepareStatement(sql);  for (String username : usernames) {  stmt.setString(1, username);  ResultSet rs = stmt.executeQuery();  // Process user data from result set  }  This code efficiently retrieves data for all users without rewriting the query.  3. Performance:   * Placeholders can boost performance by allowing the database server to pre-compile   the SQL statement. This eliminates the need to recompile the query for each  execution with different data.  4. Readability and maintainability:   * Using placeholders makes the code cleaner and easier to understand. It separates the   logic of the query from the data manipulation, leading to more maintainable and  bug-free code.  Example:  A query with hardcoded data can be difficult to decipher:  SQL  String sql = "SELECT \* FROM orders WHERE customer\_id = 123 AND order\_date =  '2023-10-26'";Using placeholders improves readability:  SQL  String sql = "SELECT \* FROM orders WHERE customer\_id = ? AND order\_date = ?";  PreparedStatement stmt = connection.prepareStatement(sql);  stmt.setInt(1, 123);  stmt.setString(2, "2023-10-26");  , placeholders are essential in JDBC programs for security, reusability, performance, and  code quality. They offer a powerful and efficient way to interact with your database  while safeguarding against vulnerabilities and enhancing maintainability. |
|  | g) | What is the need of .jar file, explain with example?  A .jar file, or Java Archive, serves several important purposes in the Java ecosystem.  Here's why it's needed:  1. Packaging and Deployment:   * Imagine a Java application with multiple class files, images, and other resources   scattered across your file system. Deploying and distributing such an unorganized  structure would be a nightmare.   * .jar files bundle all these elements into a single, neat package. This simplifies   deployment, distribution, and version control. You can easily share your application  or library with others as just one file.  Example:  You develop a library for calculating statistics. Instead of sending multiple folders and files,  you create a statistics.jar file containing all your compiled class files, algorithms, and  documentation. Users can simply download and drop this single file into their project.  2. Classpath Management:   * Java applications rely on various classes spread across different libraries and the   Java Runtime Environment (JRE). Managing individual class files becomes tedious  and error-prone.   * .jar files act as self-contained class libraries. They include a manifest file specifying   the contained classes and their dependencies. This allows the Java runtime to  automatically locate and load the required classes at runtime, simplifying  classpath management.  Example:  Your application uses a third-party library for image manipulation. Instead of adding  each library class to your classpath, you include the image-utils.jar file. The manifest in  this file tells the runtime to look for the necessary classes within the jar itself.  3. Security and Reusability:   * .jar files can be digitally signed to ensure authenticity and prevent tampering.   This is crucial for distributing libraries and applications that need to be trusted.   * By packaging classes and resources together, .jar files promote code reuse. Developers can easily share and integrate libraries without worrying about missing dependencies or conflicts.   Example:  You develop a well-tested encryption library as a signed crypto.jar file. Other developers can  incorporate this library into their projects without needing to reimplement the encryption logic.  This saves time and ensures consistent, secure encryption across applications.  In summary, .jar files are essential for packaging, deploying, managing, securing, and  reusing Java code. They provide a clean, efficient, and reliable way to distribute and  utilize Java applications and libraries |
|  | h) | Define all the constants in ResultSet interface with their values.   |  |  |  | | --- | --- | --- | | onstant | Value | Description | | TYPE\_FORWARD\_ONLY | 1000 | Indicates a forward-only result set. | | TYPE\_SCROLL\_INSENSITIVE | 1001 | Indicates a scroll-insensitive result set. | | TYPE\_SCROLL\_SENSITIVE | 1002 | Indicates a scroll-sensitive result set. | | CONCUR\_READ\_ONLY | 1003 | Indicates a read-only result set. | | CONCUR\_UPDATABLE | 1004 | Indicates an updatable result set. | |
|  | i) | WAP to print the last row in EMP table.  import java.sql.\*;  public class PrintLastRow {  public static void main(String[] args) throws SQLException {  Connection con = DriverManager.getConnection(  "jdbc:mysql://localhost:3306/system", "root", "pass");  // here sonoo is database name, root is username and password  Statement stmt = con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE,ResultSet.CONCUR\_UPDATABLE);  ResultSet rs = stmt.executeQuery("select \* from emp");  rs.last();  System.out.println(rs.getInt(1) + " " + rs.getString(2) + " " + rs.getString(3));  con.close();  }  } |
|  | j) | WAP in java to count all the rows in EMP table.  import java.sql.\*;  public class CountRows {  public static void main(String[] args) throws SQLException {  Connection con = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/dbName","root","pswd");  Statement stt = con.createStatement();  ResultSet rSet = stt.executeQuery("select count(id) from emp");  if(rSet.next())  System.***out***.println("there are "+rSet.getInt(1)+" rows in this table");  con.close();  }  } |
|  | k) | Create a stored procedure to print your college name within the console of SQL .  CREATE PROCEDURE print\_college\_name()  BEGIN  DECLARE college\_name VARCHAR(255);  SET college\_name = 'GITA';  SELECT college\_name;  END;  CALL print\_college\_name(); |
|  | l) | Define the basic steps in connecting any java application to Oracle database.  Here, Java program is loading oracle driver to establish database connection.   1. Class.forName("oracle.jdbc.driver.OracleDriver");   2) Create the connection object   |  | | --- | | The **getConnection()** method of DriverManager class is used to establish connection with the database. |   Syntax of getConnection() method   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   Example to establish connection with the Oracle database   1. Connection con=DriverManager.getConnection( 2. "jdbc:oracle:thin:@localhost:1521:xe","system","password");   3) Create the Statement object   |  | | --- | | The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database. |   Syntax of createStatement() method   1. **public** Statement createStatement()**throws** SQLException   Example to create the statement object   1. Statement stmt=con.createStatement();   4) Execute the query   |  | | --- | | 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   1. **public** ResultSet executeQuery(String sql)**throws** SQLException   Example to execute query   1. ResultSet rs=stmt.executeQuery("select \* from emp"); 3. **while**(rs.next()){ 4. System.out.println(rs.getInt(1)+" "+rs.getString(2)); 5. }   5) Close the connection object   |  | | --- | | 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   1. **public** **void** close()**throws** SQLException   Example to close connection  con.close(); |
|  | m) | Explain the different **components of a web server**.  A web server is a computer system that stores and delivers web pages to users. It is responsible for receiving requests  from web browsers, processing those requests, and sending back the appropriate web pages. Web servers are  an essential part of the World Wide Web, and they play a critical role in making sure that web pages  are accessible to users all over the world.  The different components of a web server can be broadly categorized into hardware and software  components.  Hardware Components   1. Physical Computer: The web server needs a physical computer to run on. This computer can   be a dedicated server machine, a virtual machine, or even a personal computer. The computer  must have enough processing power and memory to handle the traffic of the website.   1. Network Interface Card (NIC): The NIC is the hardware component that allows the web server   to connect to the internet. It translates data between the computer and the network cable.   1. Storage: The web server needs storage space to store the website's files, such as HTML pages, images, CSS files, and JavaScript files. This storage can be in the form of hard drives, solid-state drives, or even   cloud storage.  Software Components   1. Operating System: The operating system is the software that manages the hardware and   software resources of the web server. It provides the basic functionality that the web server  needs to run, such as memory management, file system access, and networking.   1. Web Server Software: The web server software is the core component of a web server. It is   responsible for receiving requests from web browsers, processing those requests, and sending  back the appropriate web pages. There are many different web server software programs available,  such as Apache HTTP Server, Nginx, and Microsoft IIS.   1. Scripting Languages: Scripting languages are used to generate dynamic web pages. These languages   allow web developers to create web pages that can interact with databases, process user input, and  generate customized content. Some popular scripting languages for web development include  PHP, Python, and JavaScript.   1. Database Server: A database server is used to store and manage data for a website. This data can   include things like user information, product information, and order history. The web server can  communicate with the database server to retrieve and store data as needed.   1. Application Server: An application server is used to run web applications. Web applications are   programs that run on the web server and provide functionality to users, such as e-commerce  applications, social networking applications, and content management systems.   1. Web Application Firewall (WAF): A WAF is a security device that protects web servers from attacks.   It monitors web traffic and blocks malicious requests before they can reach the web server.  These are just some of the many components that can make up a web server. The specific components  that are needed will vary depending on the size and complexity of the website. |
|  | n) | Write all the methods available in three resources of Servlet with the hierarchy.  here are all the methods available in three resources of Servlet with the hierarchy:  **\*\* Servlet Interface \*\***   |  |  | | --- | --- | | Method Name | Description | | init() | Initializes the servlet. | | service() | Services a request from a client. | | destroy() | Destroys the servlet. | | getServletConfig() | Returns the ServletConfig object for this servlet. | | getServletContext() | Returns the ServletContext object for this servlet. | | log() | Logs a message. |   **\*\* GenericServlet Abstract Class \*\***   |  |  | | --- | --- | | Method Name | Description | | init() | Initializes the servlet. | | service() | Services a request from a client. | | destroy() | Destroys the servlet. | | getServletConfig() | Returns the ServletConfig object for this servlet. | | getServletContext() | Returns the ServletContext object for this servlet. | | log() | Logs a message. | | getServletInfo() | Returns a String containing information about the servlet. | | getServletName() | Returns the name of the servlet. |   **\*\* HttpServlet Abstract Class \*\***   |  |  | | --- | --- | | Method Name | Description | | doGet() | Handles HTTP GET requests. | | doPost() | Handles HTTP POST requests. | | doHead() | Handles HTTP HEAD requests. | | doOptions() | Handles HTTP OPTIONS requests. | | doPut() | Handles HTTP PUT requests. | | doDelete() | Handles HTTP DELETE requests. | | doTrace() | Handles HTTP TRACE requests. | | service() | Services a request from a client. | | getServletConfig() | Returns the ServletConfig object for this servlet. | | getServletContext() | Returns the ServletContext object for this servlet. | | log() | Logs a message. | |
|  | o) | Write the steps to create a web application.  Creating a web application involves a series of steps, from planning and designing to developing and deploying  the application. Here's a general overview of the process:  **Planning and Defining Requirements:**  a**. Identify the Purpose**: Clearly articulate the purpose of the web application, what problem it solves, and the  value it provides to users.  b. Define Target Audience: Understand the characteristics, needs, and expectations of the target users.  c. Outline Key Features: Determine the essential features and functionalities that the web application should offer.  **Designing the User Interface (UI) and User Experience (UX):**  a. Create Wireframes: Sketch out the layout and structure of the web application's pages to visualize the user  flow.  b. Design UI Prototypes: Develop interactive prototypes to simulate the look, feel, and behavior of the application.  c. Conduct User Testing: Gather feedback from potential users to refine the UI/UX and ensure usability.  **Development and Implementation**:  a. Choose a Development Framework: Select a suitable framework like Java EE, Ruby on Rails, or Spring MVC for building the application.  b. Write Application Code: Implement the application's logic, using programming languages like Java, Ruby, or Python.  c. Integrate with Databases: Connect to databases like MySQL, PostgreSQL, or MongoDB to store and manage application data.  d. Implement Security Measures: Implement security protocols to protect the application from vulnerabilities and attacks.  **Testing and Deployment:**  a. Perform Unit Testing: Test individual components and modules of the application to ensure they function correctly.  b. Conduct Integration Testing: Verify the interactions and data flow between different components of the application.  c. Perform User Acceptance Testing: Involve real users to test the application's functionality and usability.  d. Deploy to Production Environment: Deploy the application to a web server or cloud platform for public access.  Maintenance and Post-Deployment:  a. Monitor Performance: Continuously monitor the application's performance, resource utilization, and error logs.  b. Address Bugs and Issues: Promptly address any bugs, security vulnerabilities, or performance issues that arise.  c. Implement Updates and Enhancements: Regularly update the application with new features, bug fixes, and security patches. |
|  | p) | Write different methods of sending request to Servlet.  **Methods of ServletRequest interface**  There are many methods defined in the ServletRequest interface. Some of them are as follows:   |  |  | | --- | --- | | **Method** | **Description** | | **public String getParameter(String name)** | is used to obtain the value of a parameter by name. | | **public String[] getParameterValues(String name)** | returns an array of String containing all values of given parameter name. It is  mainly used to obtain values of a Multi select list box. | | **java.util.Enumeration getParameterNames()** | returns an enumeration of all of the request parameter names. | | **public int getContentLength()** | Returns the size of the request entity data, or -1 if not known. | | **public String getCharacterEncoding()** | Returns the character set encoding for the input of this request. | | **public String getContentType()** | Returns the Internet Media Type of the request entity data, or null if not  known. | | **public ServletInputStream getInputStream() throws IOException** | Returns an input stream for reading binary data in the request body. | | **public abstract String getServerName()** | Returns the host name of the server that received the request. | | **public int getServerPort()** | Returns the port number on which this request was received. | |
|  | q) | Develop a web application where the home page contains a text field “Employee NAME” and a submit  button “Search”. Once you click on search it will display all the information of The given employees from  the table EMP (empid, ename, deptno, salary) |
|  | r) | What is ServletConfig. Explain its use, with an example.  ServletConfig is an interface in the Java Servlet API that provides access to configuration information for a  servlet. It is implemented by the servlet container and passed to the servlet during its initialization phase.  The ServletConfig object contains information such as the servlet's name, initialization parameters, and  servlet context.  Use of ServletConfig  ServletConfig is used to access configuration information that is specific to a particular servlet. This  information can be used to customize the servlet's behavior, such as by setting default values for parameters or loading resource files. For example, a servlet that displays product information could use ServletConfig to get the  name of the product database to connect to.  Example of ServletConfig  Here is an example of how to use ServletConfig to get the name of a servlet's initialization parameter:  Java  public class MyServlet extends HttpServlet {  @Override  public void init() {  String productName = getServletConfig().getInitParameter("productName");  // Use the productName parameter to initialize the servlet  }  }  In this example, the MyServlet class uses ServletConfig to get the value of the productName initialization  parameter. This parameter is specified in the servlet's deployment descriptor (web.xml) file. The value of the  parameter can be used to initialize the servlet, such as by setting the default product to display.  Benefits of Using ServletConfig  Using ServletConfig provides several benefits:   * Decouples servlets from configuration details: ServletConfig allows servlets to access configuration   information without having to hardcode it into the servlet's code. This makes servlets more reusable  and easier to maintain.   * Provides centralized configuration management: ServletConfig allows configuration information   to be managed centrally in the deployment descriptor (web.xml) file. This makes it easier to change  configuration settings without having to recompile and redeploy the servlet.   * Enables dynamic configuration: ServletConfig can be used to access configuration information that   can be changed dynamically at runtime. This can be useful for applications that need to adapt to  changing environments. |
|  | s) | What is session tracking? Explain different session tracking mechanism.  Session Tracking is a way to maintain state (data) of an user. It is also known as session management in servlet.  There are four techniques used in Session tracking:  Cookies  Hidden Form Field  URL Rewriting  HttpSession  **cookie** is a small piece of information that is persisted between the multiple client requests.  A cookie has a name, a single value, and optional attributes such as a comment, path and  domain qualifiers, a maximum age, and a version number.  How Cookie works  By default, each request is considered as a new request. In cookies technique, we add cookie with  response from the servlet. So cookie is stored in the cache of the browser. After that if request is  sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.  cookies in servlet  Hidden Form Field   1. [Hidden Form Field](https://www.javatpoint.com/hidden-form-field-in-session-tracking)   In case of Hidden Form Field **a hidden (invisible) textfield** is used for maintaining the state of an  user.  In such case, we store the information in the hidden field and get it from another servlet. This  approach is better if we have to submit form in all the pages and we don't want to depend on the  browser.  Let's see the code to store value in hidden field.   1. <input type="hidden" name="uname" value="Vimal Jaiswal">   Here, uname is the hidden field name and Vimal Jaiswal is the hidden field value.  3)URL Rewriting   1. [URL Rewriting](https://www.javatpoint.com/url-rewriting-in-session-tracking) 2. [Advantage of URL Rewriting](https://www.javatpoint.com/url-rewriting-in-session-tracking#urladv) 3. [Disadvantage of URL Rewriting](https://www.javatpoint.com/url-rewriting-in-session-tracking#urldisadv) 4. [Example of URL Rewriting](https://www.javatpoint.com/url-rewriting-in-session-tracking#urlex)   In URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource. We can send parameter name/value pairs using the following format:  url?name1=value1&name2=value2&??  A name and a value is separated using an equal = sign, a parameter name/value pair is separated from another parameter using the ampersand(&). When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server. From a Servlet, we can use getParameter() method to obtain a parameter value.  HttpSession interface   1. [HttpSession interface](https://www.javatpoint.com/http-session-in-session-tracking) 2. [How to get the HttpSession object](https://www.javatpoint.com/http-session-in-session-tracking#httpsessionhow) 3. [Commonly used methods of HttpSession interface](https://www.javatpoint.com/http-session-in-session-tracking#httpsessionmethod) 4. [Example of using HttpSession](https://www.javatpoint.com/http-session-in-session-tracking#httpsessionex)   In such case, container creates a session id for each user.The container uses this id to identify the  particular user.An object of HttpSession can be used to perform two tasks:   1. bind objects 2. view and manipulate information about a session, such as the session identifier, creation   time, and last accessed time. |
|  | t) | Explain the working of JSP lifecycle.  he Lifecycle of a JSP Page  The JSP pages follow these phases:   * Translation of JSP Page * Compilation of JSP Page * Classloading (the classloader loads class file) * Instantiation (Object of the Generated Servlet is created). * Initialization ( the container invokes jspInit() method). * Request processing ( the container invokes \_jspService() method). * Destroy ( the container invokes jspDestroy() method).   Note: jspInit(), \_jspService() and jspDestroy() are the life cycle methods of JSP.  How JSP is converted into Servlet  As depicted in the above diagram, JSP page is translated into Servlet by the help of JSP translator.  The JSP translator is a part of the web server which is responsible for translating the JSP page into  Servlet. After that, Servlet page is compiled by the compiler and gets converted into the class file.  Moreover, all the processes that happen in Servlet are performed on JSP later like initialization,  committing response to the browser and destroy. |
|  | u) | Explain the use and syntax of five JSP tags(JSP Elements).  **1. Directive Tag**  **Use:** Directive tags are used to provide instructions to the JSP compiler. They are used to specify the  JSP version, include files, and set attributes for the JSP page.  **Syntax:**  Java  <%@ directive attribute="value" %>  **2. Declaration Tag**  **Use:** Declaration tags are used to declare variables, methods, and class definitions within a JSP page.  **Syntax:**  Java  <%! declaration %>  <%!  private String message = "Hello, World!";  %>  **. Scriptlet Tag**  **Use:** Scriptlet tags are used to embed Java code directly into a JSP page. They are used to perform  dynamic operations, such as accessing variables, manipulating data, and calling methods.  **Syntax:**  Java  <% scriptlet code %>  Java  <%  String name = request.getParameter("name");  out.println("Welcome, " + name + "!");  %>  **4. Expression Tag**  **Use:** Expression tags are used to evaluate Java expressions and insert the result into the JSP page.  They are more concise and efficient than scriptlet tags.  **Syntax:**  Java  <%= expression %>  Java  <%= message %>  Use code with caution. [Learn more](https://bard.google.com/faq#coding)  content\_copy  **5. Action Tag**  **Use:** Action tags are used to perform specific actions, such as forwarding to another JSP page, including a fragment, or accessing a bean. They are more structured and reusable than scriptlet tags.  **Syntax:**  Java  <jsp:action action="actionName" attributeName="attributeValue" />  Java  <jsp:forward page="success.jsp">  <jsp:param name="message" value="Your request has been processed." />  </jsp:forward> |
|  | v) | What is ORM? Explain different frameworks in brief.  object/relational mapping is the automated (and transparent) persistence of objects in a Java application to the tables in a relational database, using  metadata that describes the mapping between the objects and the database. ORM, in essence, works by (reversibly) transforming data from one representation to another.  ORM is also called as object role modeling/object relational mapping.      Different ORM frameworks offer varying features and cater to different programming languages and database systems. Here are some popular ORM frameworks:   * Hibernate: A mature and widely used ORM framework for Java, supporting various databases. * Spring Data JPA: A popular ORM framework integrated with Spring, simplifying data access in Spring applications. * MyBatis: A flexible and lightweight ORM framework for Java, offering more control over SQL queries. * Entity Framework: A .NET framework for mapping objects to tables in relational databases.   1. Enterprise JavaBeans Entity Beans==  2. Java Data Objects  3. Top Link  4. Spring DAO  5. Hibernate  6. And many more  EJB Entity Beans have been largely superseded by JDOs and ORMs like Hibernate.  JDOs provide a standardized approach with POJO support,  while TopLink is a powerful commercial ORM.  Spring DAO focuses on separating data access logic,  while Hibernate offers a feature-rich and open-source ORM solution.  **Optional**  Object-relational mapping (ORM) is a programming technique that connects object-oriented programming  languages with relational databases. It bridges the gap between the object-oriented world of software  development and the relational world of data storage. ORM frameworks provide a set of tools and abstractions that simplify the process of working with relational databases using object-oriented programming languages.  ORM frameworks typically provide the following features:   * Mapping between objects and database tables: ORM frameworks automatically map object properties to database columns, allowing developers to work with objects instead of writing SQL queries directly. * Data access abstraction: ORM frameworks hide the complexity of SQL queries and database interactions, allowing developers to focus on the business logic of their applications. * Change tracking and persistence management: ORM frameworks track changes made to objects and automatically update the corresponding database records, ensuring data consistency. * Querying and manipulation of data: ORM frameworks provide object-oriented methods for querying and manipulating data, making it easier to work with complex data structures. * databases.   1. Enterprise JavaBeans (EJB) Entity Beans  EJB Entity Beans are a persistence layer technology within the EJB specification. They provide a way to map Java objects to relational databases, simplifying data access and management. Key features include:   * Automatic persistence: EJB containers manage persistence logic, saving and retrieving entity beans to and from the database. * Transaction management: Transactions are automatically handled, ensuring data consistency. * Relationships: EJBs support various relationships between entities, like one-to-many and many-to-many. * Life cycle management: The EJB container manages the entity bean lifecycle, including creation, initialization, and removal.   EJB Entity Beans are a mature technology but have been largely superseded by Java Data Objects (JDOs) and Object/Relational Mapping (ORM) frameworks like Hibernate due to their complexity and vendor lock-in.  2. Java Data Objects (JDOs)  JDOs are a standard specification for representing persistent data in Java applications. They offer a simpler and more portable alternative to EJB Entity Beans. Key features include:   * Standard annotations: JDOs use annotations to define persistence behavior, making code cleaner and less dependent on specific vendors. * POJO support: JDOs can work with Plain Old Java Objects (POJOs), simplifying data access logic. * Multi-vendor support: JDOs are not tied to a specific vendor, allowing for greater flexibility and choice.   However, JDOs have seen less widespread adoption compared to ORMs like Hibernate.  3. TopLink  TopLink is a popular ORM framework developed by Oracle. It provides powerful features for mapping Java objects to relational databases, including:   * Advanced mapping capabilities: TopLink supports various mapping strategies, inheritance, and complex relationships. * Performance optimization: TopLink offers caching and query optimization techniques for improved data access speed. * Integration with other Oracle products: TopLink seamlessly integrates with other Oracle technologies like WebLogic and Oracle Database.   Although robust, TopLink is a commercial product with associated licensing costs, making it less accessible to some projects.  4. Spring DAO  Spring DAO (Data Access Object) is a design pattern and framework within the Spring ecosystem. It provides a clean way to separate data access logic from business logic, promoting modularity and maintainability. Key features include:   * Template method pattern: Spring DAO uses the template method pattern to simplify data access code. * Abstraction of data access layers: Developers can work with different data access technologies (JDBC, JDO, ORMs) without changing their application code. * Integration with other Spring components: Spring DAO seamlessly integrates with other Spring features like transaction management and dependency injection.   Spring DAO is a popular choice due to its ease of use, flexibility, and integration with other Spring technologies.  5. Hibernate  Hibernate is a widely used and open-source ORM framework. It offers powerful and flexible features for mapping Java objects to relational databases, including:   * Annotation-based configuration: Hibernate supports defining persistence behavior through annotations, simplifying configuration. * Lazy loading: Hibernate loads data only when needed, improving performance. * Query language: Hibernate provides its own query language (HQL) similar to SQL, offering abstraction and flexibility. * Large community and ecosystem: Hibernate has a large and active community, providing extensive documentation, tutorials, and plugins.   Hibernate is a popular choice due to its open-source nature, powerful features, and ease of use. |
|  | w) | Write all the methods available in three resources of Servlet with the hierarchy.  here are three main resources in the Servlet class hierarchy, each offering different sets of methods:  1. javax.servlet.Servlet Interface:   * Abstract methods:   + init(ServletConfig config): Called when the servlet is first initialized.   + service(ServletRequest req, ServletResponse resp): Called for every client request to the servlet.   + destroy(): Called when the servlet is destroyed. * Optional methods:   + getServletConfig(): Returns the ServletConfig object associated with the servlet.   + getServletContext(): Returns the ServletContext object associated with the servlet.   + log(String msg): Logs a message to the servlet container's log file.   2. javax.servlet.http.HttpServlet Class:   * Extends: javax.servlet.Servlet interface. * Additional methods:   + doGet(HttpServletRequest req, HttpServletResponse resp): Handles GET requests specifically.   + doPost(HttpServletRequest req, HttpServletResponse resp): Handles POST requests specifically.   + doPut(HttpServletRequest req, HttpServletResponse resp): Handles PUT requests specifically.   + doDelete(HttpServletRequest req, HttpServletResponse resp): Handles DELETE requests specifically.   + doHead(HttpServletRequest req, HttpServletResponse resp): Handles HEAD requests specifically.   + doOptions(HttpServletRequest req, HttpServletResponse resp): Handles OPTIONS requests specifically.   + service(ServletRequest req, ServletResponse resp) (overrides Servlet.service): Dispatches requests to the appropriate doXXX method based on the HTTP method used.   3. javax.servlet.http.HttpSession Interface:   * Separate from Servlet class hierarchy: Accessible via HttpServletRequest.getSession(). * Methods for managing sessions:   + setAttribute(String name, Object value): Stores an attribute in the session.   + getAttribute(String name): Retrieves an attribute from the session.   + removeAttribute(String name): Removes an attribute from the session.   + invalidate(): Invalidates the session.   + getMaxInactiveInterval(): Gets the maximum inactive interval for the session.   + setMaxInactiveInterval(int seconds): Sets the maximum inactive interval for the session.   + isNew(): Checks if the session is new (not previously used by the client).   + getServletContext(): Returns the ServletContext associated with the session.   Hierarchy:  javax.servlet.Servlet  |  +-- javax.servlet.http.HttpServlet  |  +--- javax.servlet.http.HttpSessio |
|  | x) | Write the steps to create a RMI application.  1  Creating a Remote Method Invocation (RMI) application involves several steps, including:  **Define the Remote Interface:** Create a Java interface that defines the methods you want to expose remotely.  This interface extends java.rmi.Remote and declares methods that throw RemoteException.  **Implement the Remote Interface**: Create a class that implements the remote interface, providing the actual implementation for the remote methods. This class must also extend java.rmi.server.UnicastRemoteObject.  **Create the Remote Object**: Create an instance of the remote implementation class and export it using the  RMIRegistry. This process makes the remote object available for clients to connect to.  **Start the RMI Registry**: Start the rmiregistry service to provide a central registry for locating remote objects.  **Create the RMI Client**: Create a client application that connects to the RMI registry and obtains a stub for the  remote object. The stub acts as a proxy for the remote object, allowing the client to invoke remote methods.  **Run the Client and Server**: Run the client application and the server application. The client should be able to  connect to the server and invoke remote methods. |
|  | y) | Explain Servlet lifecycle with suitable diagram.  e web container maintains the life cycle of a servlet instance. Let's see the life cycle of the  servlet:   1. Servlet class is loaded. 2. Servlet instance is created. 3. init method is invoked. 4. service method is invoked. 5. destroy method is invoked.   Life cycle of a servlet  1) Servlet class is loaded  The classloader is responsible to load the servlet class. The servlet class is loaded when the first  request for the servlet is received by the **web container**.  2) Servlet instance is created  The web container creates the instance of a servlet after loading the servlet class. The servlet instance  is created only **once i**n the servlet life cycle.  3) init method is invoked   |  | | --- | | The web container calls the init method only once after creating the servlet instance. The init  method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet  interface. Syntax of the init method is given below: |  1. **public** **void** init(ServletConfig config) **throws** ServletException   4) service method is invoked  The web container calls the service method each time when request for the servlet is received.  If servlet is not initialized, it follows the first three steps as described above then calls the service  method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once.  The syntax of the  service method of the Servlet interface is given below:   1. **public** **void** service(ServletRequest request, ServletResponse response) 2. **throws** ServletException, IOException   5) destroy method is invoked  The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc. The syntax of the destroy method of the Servlet interface is given below:   1. **public** **void** destroy() |
|  | z) | Develop a web application where the home page contains a text field “Employee NAME” and a submit button  “Search”. Once you click on search it will display all the information of the given employees from the table EMP  (empid,ename,deptno,salary) |
| Part–III | | |
|  | Long Answer type Questions | |
| 03 | a) | What is the need of a database driver? Differentiate API document and driver.  A database driver is a software component that acts as a bridge between an application and a database  management system (DBMS). It allows the application to communicate with the database, send queries, and  retrieve data. Without a database driver, an application cannot connect to a database or perform any database  operations.  **Need of a Database Driver**   1. Standardization: Database drivers provide a **standardized interface** for applications to interact with   different DBMSs, regardless of their underlying implementation or data storage format. This  standardization simplifies application development and reduces the need for developers to learn multiple database-specific protocols.   1. Translation: Database drivers translate **application-level requests into database-specific commands** that   the DBMS can understand. They handle the complexities of database communication, such as data  **encoding, error handling, and transaction management, so** that developers can focus on the application logic.   1. Performance Optimization: Database drivers are optimized for specific DBMSs and can leverage the DBMS's native capabilities to improve query performance and data access efficiency. They can also handle various data   types and data manipulation operations efficiently.  **API Document vs. Driver**  An API document is a set of **specifications that describes the methods, functions, and classes available in an**  **AP**I. It provides detailed **information about the API's structure, usage, and behavior**. A database driver, on the  other hand, is an **implementation of an API that allows applications to interact with a specific DBMS**.  It **translates API calls into database-specific commands** and handles the underlying communication with the  DBMS.  In essence, an **API document is a blueprint for the API**, while a **database driver is a tool that brings the API to**  **life** and allows applications to utilize its features.  Summary Table   |  |  |  | | --- | --- | --- | | Feature | API Document | Database Driver | | Purpose | Describes the API's structure, usage, and behavior | Translates API calls into database  -specific commands | | Functionality | Provides no actual implementation | Provides an implementation of the  API for a specific DBMS | | Role in Application | Guides developers in using the API | Enables applications to  interact with the DBMS | | Focus | API specifications and usage | Database communication and data access | |
|  | b) | WAP to display the name of the employee who is getting the highest salary.  import java.sql.\*;  public class PrintMaxSalary {  public static void main(String[] args) throws SQLException {  Connection con = DriverManager.getConnection( "jdbc:mysql://localhost:3306/sonoo", "root", "pasword");  Statement stmt = con.createStatement();  ResultSet rs = stmt.executeQuery("select name from emp where salary = (select max(salary) from emp)");  if(rs.next())  System.out.println(rs.getString(1)+ " is getting highest salary");  con.close();  }  } |
| 04 | a) | Draw the basic architecture of a web application. |
|  | b) | Write the features of Servlet.  features of servlet  **Better performance:**Because it creates a thread for each request not process (like CGI).  **Portability:**Because it uses java language and java is robust language.  **Robust:**Servlet are managed by JVM so no need to worry about memory leak, garbage collection etc.  **Secure:**Because it uses java language and java is a secure language. Java have automatic  garbage collection mechanism and a lack of pointers protect the servlets from memory management  problems.  **Inexpensive** There are number of free web servers available for personal use or for commercial  purpose. Mostly web server are very costly. So by using free web server you can reduce project  development price.  **Extensibility** The servlet API is designed in such a way that it can be easily extensible. Servlets being  written in Java, can be extended and polymorphed into the objects that suits the user requirement.  **Efficiency** Servlets invocation is highly efficient as compared to any CGI programs.  **Integration** Servlets are tightly integrated with the server. Servlet can use the server to translate the file paths, check authorization, perform logging and MIME type mapping etc.  **Persistent:** Servlets remain in memory until explicitly destroyed. This helps in serving several  incoming requests. Servlets establishes connection only once with the database and can handle  several requests on the same database.  **Server Independent:** Servlets are compatible with any web server available today.  **Protocol Independent:** Servlets can be created to support any protocols like FTP commands, Telnet  sessions, NNTP newsgroups, etc. It also provides extended support for the functionality of HTTP  protocol.  **Fast:** Since servlets are compiled into bytecodes, they can execute more quickly as compared to  other scripting languages. The bytecode compilation feature helps servlets to give much better  performance. In addition, it also provides advantage of strong error and type checking. |
| 05 | a) | Explain the RMI architecture.    Here RRL stands for remote reference layer which manages the references by the client to the remote object  The **RMI** (Remote Method Invocation) is an API that provides a mechanism to create distributed  application in java. The RMI allows an object to invoke methods on an object running in another JVM.  The RMI provides remote communication between the applications using two objects *stub* and *skeleton*.  Stub  It resides at the client side and represents the remote object,it does the following tasks:   1. It initiates a connection with remote Virtual Machine (JVM), 2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM), 3. It waits for the result 4. It reads (unmarshals) the return value or exception, and 5. It finally, returns the value to the caller   skeleton  The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it.  When the skeleton receives the incoming request, it does the following tasks:  It **reads the parameter** for the remote method  It **invokes the method on the actual remote object**, and  It **writes and transmits (marshals) the result to the caller**. |
|  | b) | Explain the working principle of 4-types of JDBC drivers.  DBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC  drivers:  JDBC-ODBC bridge driver  Native-API driver (partially java driver)  Network Protocol driver (fully java driver)  Thin driver (fully java driver)  JDBC-ODBC bridge driver   |  | | --- | | The JDBC-ODBC bridge driver uses **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 thin driver. |   bridge driver  Native-API 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. |   Native-API driver  3) Network Protocol driver  The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or  indirectly into the vendor-specific database protocol. It is fully written in java.  Network Protocol driver   Thin driver   |  | | --- | | The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language. |   Thin driver |
| 06 | a) | WAP in java to insert row into EMP table using PreparedStatement.  import java.sql.\*;  public class PreparedInsertion {  public static void main(String[] args) throws SQLException {  Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/sonoo", "root", “pswd");  PreparedStatement pst = con.prepareStatement("insert into emp values(?,?,?)");  pst.setInt(1, 11);  pst.setString(2, "Akash");  pst.setInt(3, 23);  int i = pst.executeUpdate();  System.out.println(i+" records inserted");  }  } |
|  | b) | WAP in java to delete a row from to EMP table using  PreparedStatement correspond to an employee whose salary is 10000  import java.sql.\*;  public class DeleteRow {  public static void main(String[] args) throws SQLException {  Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/sonoo", "root",  "password");  PreparedStatement pst = con.prepareStatement("delete from emp where salary = ?");  pst.setInt(1,10000);  int i = pst.executeUpdate();  System.out.println(i+" records deleted");  }  } |
| 7) | a) | WAP in java to read the third row from top in EMP table  import java.sql.\*;  public class Top\_3rd {  public static void main(String[] args) throws SQLException  {  Connection con = DriverManager.getConnection(  "jdbc:mysql://localhost:3306/sonoo", "root", "password");  Statement stmt = con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE,ResultSet.CONCUR\_UPDATABLE);  ResultSet rs = stmt.executeQuery("select \* from emp");  rs.absolute(3);  System.out.println(rs.getInt(1) + " " + rs.getString(2) + " " + rs.getString(3));  con.close();  }  } |
|  | b) | WAP in java to read the fifth row from bottom in EMP table  package eclipse;  import java.sql.\*;  public class Bottom\_5th {  public static void main(String[] args) throws SQLException  {  Connection con = DriverManager.getConnection(  "jdbc:mysql://localhost:3306/sonoo", "root", "password");  Statement stmt = con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE,ResultSet.CONCUR\_UPDATABLE);  ResultSet rs = stmt.executeQuery("select \* from emp");  rs.last();  int counter = 5;  while(counter>1) {  rs.previous();  counter--;}  System.out.println(rs.getInt(1) + " " + rs.getString(2) + " " + rs.getString(3));  con.close();  }  } |
| 8) | a) | WAP in java to call a stored procedure present in Oracle11g using java program to insert a row into  EMP table.  import java.sql.\*;  import oracle.jdbc.OracleCallableStatement; // Import the Oracle CallableStatement class  public class InsertEmpStoredProcedure {  public static void main(String[] args) throws Exception {  // Database connection details  String url = "jdbc:oracle:thin:@<hostname>:<port>/<database>";  String username = "<username>";  String password = "<password>";  // Stored procedure name and parameters  String storedProcedureName = "INSERT\_EMP\_SP"; // Replace with your actual stored procedure name  int empId = 1234; // Replace with your actual employee ID  String empName = "John Doe"; // Replace with your actual employee name  String department = "IT"; // Replace with your actual department  // Connect to the database  Connection connection = DriverManager.getConnection(url, username, password);  // Prepare the callable statement  CallableStatement callableStatement = connection.prepareCall("{call " + storedProcedureName + "(?, ?, ?)}");  // Set the IN parameters  callableStatement.setInt(1, empId);  callableStatement.setString(2, empName);  callableStatement.setString(3, department);  // Execute the stored procedure  callableStatement.execute();  // Close the resources  callableStatement.close();  connection.close();  System.out.println("Row inserted successfully into the EMP table!");  }  } |
|  | b) | WAP in java to print all the rows in EMP table in reverse order.  package eclipse;  import java.sql.\*;  public class PrintRev {  public static void main(String[] args) throws SQLException {  Connection con = DriverManager.getConnection(  "jdbc:mysql://localhost:3306/sonoo", "root", "passwd");  Statement stmt = con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE,ResultSet.CONCUR\_UPDATABLE);  ResultSet rs = stmt.executeQuery("select \* from emp");  rs.last();  while(rs.previous())  {  System.out.println(rs.getInt(1) + " " + rs.getString(2) + " " + rs.getString(3));}  con.close();  }  } |
| 9) | a) | Explain the use of the tag<welcome-file-list>and<welcome- file>.  The <welcome-file-list> and <welcome-file> tags are used in the web.xml file of a web application deployed on a container  like Tomcat or GlassFish. They work together to specify which file the web server should serve when a user accesses the  application **without providing a specific file name in the URL.**  Here's how they work:   * <welcome-file-list>: This element defines a list of files that can act as the "welcome file." It's like a menu of   potential landing pages for your application.   * <welcome-file>: Each element within the <welcome-file-list> element specifies a single file that can be the   welcome file. You can have multiple <welcome-file> elements listed, defining a priority order.  How the server chooses the welcome file:   1. When a user accesses the application without specifying a file name in the URL (e.g., <http://localhost:8080/myapp>),    the server consults the <welcome-file-list> element in the web.xml file.   1. The server checks the first <welcome-file> element in the list. If that file exists, it serves it as the landing page of your application. 2. If the first <welcome-file> doesn't exist, the server moves on to the next one in the list, and so on. 3. If none of the <welcome-file> elements exist, the server typically returns a 404 Not Found error.   Benefits of using <welcome-file-list> and <welcome-file>:   * Convenience: Users don't need to remember or type the full file name of your landing page. * Flexibility: You can define multiple potential landing pages and prioritize them. * Common practice: Using these tags is a standard practice in web application development, making it easier for   others to understand your application's structure.  **extra:**   * The file extension is typically not included in the <welcome-file> element. For example, if you want index.html to be the welcome file, you would simply write <welcome-file>index</welcome-file>. * Some containers have default welcome files even if you don't define any in your web.xml. You can check the documentation for your specific container to learn more. * Make sure your <welcome-file> entries are accessible to the user accessing the application. |
|  | b) | Explain client server architecture with suitable diagram.   * **Client:** When we talk the word **Client**, it mean to talk of a person or an organization   using a particular service. Similarly in the digital world a **Client** is a computer (**Host**)  i.e. capable of receiving information or using a particular service from the service  providers (**Servers**).   * **Servers:** Similarly, when we talk the word **Servers**, It mean a person or medium   that serves something. Similarly in this digital world a **Server** is a remote computer  which provides information (data) or access to particular services.  So, its basically the **Client** requesting something and the **Server** serving it as long as its  present in the database.    **How the browser interacts with the servers ?** There are few steps to follow to interacts with the servers and client.   * User enters the **URL**(Uniform Resource Locator) of the website or file. The Browser   then requests the **DNS**(DOMAIN NAME SYSTEM) Server.   * **DNS Server** lookup for the address of the **WEB Server**. * **DNS Server** responds with the **IP address** of the **WEB Server**. * Browser sends over an **HTTP/HTTPS** request to **WEB Server’s IP** (provided by   **DNS server**).   * Server sends over the necessary files of the website. * Browser then renders the files and the website is displayed. This rendering is done   with the help of **DOM** (Document Object Model) interpreter, **CSS** interpreter and  **JS Engine** collectively known as the **JIT** or (Just in Time) Compilers.    **Advantages of Client-Server model:**   * Centralized system with all data in a single place. * Cost efficient requires less maintenance cost and Data recovery is possible. * The capacity of the Client and Servers can be changed separately.   **Disadvantages of Client-Server model:**   * Clients are prone to viruses, Trojans and worms if present in the Server or uploaded   into the Server.   * Server are prone to Denial of Service (DOS) attacks. * Data packets may be spoofed or modified during transmission. * Phishing or capturing login credentials or other useful information of the user are   common and **MITM(Man in the Middle) attacks are common**. |
| 10) | a) | Explain Hibernate architecture with suitable neat diagram.   Configuration: Configuration is a class which is present in **org.hibernate.cfg** package. It activates Hibernate framework. It reads both configuration file and mapping files.   * + It activate Hibernate Framework  Configuration cfg=new Configuration();  * + It read both cfg file and mapping files  cfg.configure();  * + It checks whether the config file is syntactically correct or not.   + If the config file is not valid then it will throw an exception. If it is valid then it creates a   meta-data in memory and returns the meta-data to object to represent the config file. SessionFactory:SessionFactory is an Interface which is present in org.hibernate package and it is used to createSession Object.  * + It is immutable and thread-safe in nature.   + buildSessionFactory() method gathers the meta-data which is in the cfg Object.   + From cfg object it takes the JDBC information and create a JDBC Connection.  SessionFactory factory=cfg.buildSessionFactory(); **Session:**   * + Session is an interface which is present in org.hibernate package. Session object is created based upon SessionFactory object i.e. factory.   + It opens the Connection/Session with Database software through Hibernate Framework.   + It is a light-weight object and it is not thread-safe.   + Session object is used to perform CRUD operations.  Session session=factory.buildSession();Transaction:  * + Transaction object is used whenever we perform any operation and based upon that operation there is some change in database.   + Transaction object is used to give the instruction to the database to make the changes that happen because of operation as a permanent by using commit() method.  Transaction tx=session.beginTransaction();  * + **tx.commit();**  example Query:  * + Query is an interface that present inside org.hibernate package.   + A Query instance is obtained by calling Session.createQuery().   + This interface exposes some extra functionality beyond that provided by Session.iterate()   and Session.find():   1. A particular page of the result set may be selected by calling setMaxResults(), setFirstResult(). 2. Named query parameters may be used.  Query query=session.createQuery(); |
|  | b) | Develop a web application where the home page contains four text fields (ID, NAME, DEPT Number, SALARY) and a  Submit button “Create”. By clicking on the Create button a new row will inserted in the EMP table. |