① Demonstrate how a child class can access a protected member of its parent class within the same package. Explain with example what happens when the child class is in a different package?

Ans:

In Java protected members of a class are accessible:

- Within the same package (like default access)
- In subclass, even if those subclass are in different packages.

- Child in the same package:-

Parent.Java:

```
package mypackage;
public class parent{
    protected String message = "Hellow from parents!";
}
```

Child.Java:-

```
Package mypackage;
public class child extends Parents{
    public void showMessage()
    { System.out.println(message);
}
}
```

Main.java.

```
package mypackage;
public class Main{
public static void main (String [] args){
  Child child = new child();
  child . showMenage();
}
```

Different package:-

Parent.java:

```
package parentpkg;
public class Parentpkg
{ protected string = "Love You";
}
```

Child.java:

```
Package childpkg;
import parentpkg.Parent;
public class child extends parent{
  public void show(){
    sout (manage); //Accesible
  }
public void test()
  { parent p = new Parent() //not Accesible;
}
```

Main.Java:-

```
package childpkg;
public class Main{
 public static void main (String [] args)
  { child c = new child;
    c.show();
  }
}.
```

(2) Compare abstract classes and interfaces in terms of multiple inheritence. When would you use that?

Ans:

Interface:

A class can implement multiple inhe. Interfaces. Java do not support multiple inheritence but allow multiple inheritence for interfaces

Ex:

```
interface A { void method A(); }
interface B { void method B(); }
class C implements A, B { .. }.
```

Abstract class:

A class can only one abstract class. (single inhe

Ex:

```
abstract class Aba { abstract void method(); }.
class B extends Aba { .. }.
```

Use an interface when:—

. We need multiple inheritence of type. A class an implement multiple interfaces but extend only one class.

When to use an abstract class:

- We can use it to share common code among related class eg. Abstract classes can have concrete methods with default behavior.
- abstract can have constructor but interface cannot.

(3)

Encapsulation means hiding internal data of a class and allowing access only through controlled methods (getter / setter)

Data security:-

- Variable that are marked private cannot be accessed or modified directly from outside the class..
- Only class itself decide how these variables are set or changed.

Ensure data Integrity:-

Setters like setAccountNumber() and setInitialBalance() include validation checks:

- If someone tries to set a null, empty or negative value - the class reject it.
- This protect the object from being put into an invalid state.

Ex:

```
Public class BankAccount{
private String accountNumber;
private double balance;

public void setAccountNumber(String accountNumbe){
if(accountNumber == NULL || accountNumber.trim().isEmpty())
{ sout(" Invalid account Number."); return;
}
this.accountNumber = accountNumber;
}
Public Void setInitialBalance(double fk){
if(fk < 0){
    Sout(" Initial Balance cannot be negative:");
    return;
}
thi. balance = "fk;
}
Public String getAccountNumber(){ return Acountnumber;}
Public Duble getBalance(){return balance;}
}
```

```
public class BankTest {
    public static void main(String[] args) {
        BankAccount acc = new BankAccount();
        acc.setAccountNumber("ABC1234");
        acc.setInitialBalance(5000);
        System.out.println("Account " + acc.getAccountNumber());
    }
}
```

~~Ans: to —— 4~~

Ans: to the ques: NO-04

4)

① Find the kth smallest element.

```
import java.util.*;
public kthSmallest{
    public static void main (String[] args){ }
ArrayList<Integer> list = new ArrayList<>();
Scanner sc = new Scanner (System.in);
int n = sc.nextInt();
for (int i=0; i<n; i++) list.add(sc.nextInt());
int k = sc.nextInt();
if(k>0 && k<=list.size()){
Collections.sort(list);
Sout(k+"th element"+list.get(k-1);
}
}.
```

New ArrayList<>(Arrays.asList(7,2,9,8,1,6));

(11) Treemap to store word frequency :-

```
import java.util.*;
public class WordFrequency{
  public static void main (String[] args){
  String text = "Java is not better than Cpp";
Tree Map<String, Integer> map = new TreeMap<>();
String[] words = text.toLowerCase().split("\\W+");  // \W+ -> non word
for (String word : words)
  {if (word.isEmpty()) continue;
    map.put(word, map.getOrDefault(word,0)+1);
  }
For (Map.Entry<String, Integer> entry : map.entrySet())
{sout (entry.getKey() + " => " + entry.getValue());
}}}
```

(V)
```
import java.util.*;
public class LinkedListEquality{
  public static void main(String[] args).
  {
    LinkedList<String> list1 = new LinkedList<>(Arrays.asList
                                  ("A","B","C"));
LinkedList<String> list2 = new LinkedList<>(Arrays.asList("A","B","C"));
System.out.println("list1 equals list2 ? "+list1.equals(list2));
  }
}.
```

Ans: to the ques NO-05.

RegistrarParking.java

```
private final string carNumber;
public RegistrarParking (String carNumber){
    this.carNumber.=carNumber;
}
public string getcarNumber (){ return carNumber;
}
}
```

ParkingPool.java:—

```
import java.util.LinkedList;
import java.util.Queue;
public class ParkingPool{
private final Queue<RegistrarParking> parkingQueue=
new LinkedList<>();
public synchronized void requestParking (RegistrarParking
car)
{ parkingQueue.add(car);
 system.out.println("Car" + car.getCarNumber()+
"requested parking.");
 notifyAll();
}
```

```
Public synchronized RegistrarParking getNextCar(){
 while (ParkingQueue.isEmpty()){
   try { wait();
    }catch (InterruptedException e){
    Thread.currentThread().interrupt();
   }
  }
  return parkingQueue.poll();
}
```

ParkingAgent.java:-

```
public class ParkingAgent extends Thread{
  private final int agentId;
  private final ParkingPool pool;
  public ParkingAgent(int agentId, ParkingPool pool)
     { this.agentId = agentId;
       this.pool = pool;
     }
  public void run(){
     while(true)
      { registrarParking car = Pool.getNextCar();
      if (Car1 = null ){
            System.out.println("Agent "+ agentId +
 " Parked car" + Car.getCarNumber() + ".");
```

```
try{ Thread.sleep(500);
catch (InterruptedException e)
    { Thread.CurrentThread().interrupt();
    }
}
}
```

MainClass.Java

```
public class Mainclass{
public static void main (String[] args){
parkingPool pool = new ParkingPool();
ParkingAgent agent1 = new ParkingAgent(1, pool);
ParkingAgent agent2 = new ParkingAgent(2, pool);
agent1.start();
agent2.start();
String[] carNumber = {"ABC 123", "King"};
for (String carNum : CarNumbers){
new Thread(()->{
RegisterParking request = new RegisterParking
(CarNum); pool.requestParking(request);}).start
```

```
try {
    Thread.sleep(100);
}
catch (InterruptedException e){
Thread.currentThread().interrupt();
}
}
}
```

Ans to the ques: NO-06

Java provides two primary APIs for handling XML data:

1. DOM (Document Object Model) parser.
2. SAX (Simple API for ~~XML~~ XML) Parser.

Both are part of Java's standard ~~bi~~ libraries and serve different purposes depending on the structure and size of the XML file.

DOM parser:

How it works:-

1. Loads entire XML document into memory and represents it as a tree structure.
2. Allows random access and Modification of any part of the XML document.

Key features:

- Reads and stores the entire XML in memory as a tree
- Easy to Navigate, query, and update nodes.
- Good for smaller XML files.

SAX Parser.

How it Works:-

- It reads the XML document line by line and triggers events during parsing.
- Does not load the whole document into memory.

Key features:

- Faster and more memory efficient.
- Cannot modify the XML.
- Suitable for large XML files or streaming scenarios.

* Comparison Table: DOM Vs SAX:-

Feature	Dom Parser	SAX Parser
Memory usage	High- Load Full XML into Memory	Low- Process XML line by line.
Processing speed	slower - for larger file	Faster for large file.
Access Pattern	Random Access.	sequential only
Modification	Can Modify the XML structure	cannot modify the XML.
Ease of Use.	Easier to use and intuitive tree Model.	More complex due to event handling
Use cases	small to medium XMLs, needing updates	Large XMLs, read-only, streaming.

Example of SAX over DOM!—

§ Imagine ~~you~~ we are building a log monitoring tool that scans a massive 2GB ~~XML~~ XML log file to extract specific error messages.

Why SAX is preferred!—

- We don't need to load the full 2GB into memory
- We only care about specific tags, so SAX can trigger events when those tags are encountered.
- It ~~processing~~ processes the file line-by-line, using very little memory.

∴ SAX provides better performance and avoids memory overflow, which would be a big problem with DOM for such large files.

Ans: the ques: No-07.

How the virtual Dom in React Improves performance:

The virtual DOM is a core concept in React that dramatically improves UI update performance by minimizing direct interaction with the real DOM, which is slow and expensive to update.

Traditional DOM before Virtual DOM:—

- Any changes to the UI results in direct manipulation of the real DOM.
- This causes frequent reflows and repaints, which are computationally expensive.
- Updating even a small part of the UI may cause the entire page to re-render.

Virtual DOM in React:—

- React uses a virtual DOM, which is an in-memory representation of the actual DOM.
- When the state or props change:
  1. A new Virtual DOM tree is created.
  2. It is compared with the previous VDOM tree.
  3. Minimal changes are calculated.

4. Only those changes are applied to the real DOM.

Diffing Algorithm (Reconciliation):-

React uses a highly efficient diffing Algorithm based on the following assumptions:

1. Two elements of different types will produce different trees.
2. The developer can help by using key props for lists to track elements.

How it works:

- React performs a d.f.s traversal of the old and new VDOM trees.
- Compares:
  - Type of elements (div vs span - replace).
  - Attribute/props (e.g., class, style - patch)
  - children. (recursively diff).

If a node is unchanged, it skips re-rendering that subtree.

Simple Example:

```
Function Counter({count}){
return (
    <div>
        <h1> Counter </h1>
        <p> {count} </p>
    </div>
);
}
```

Initial render with count = 0.
After count change to 1.

What happens:-

1. React creats a new VDOM for 1 .
2. Compares it to the previous VDOM: 0 .
3. Detects a change only in the text node inside
4. Updates only that node in the real DOM, instead of re-redering the whole .

VDOM vs DOM

Features	Traditional DOM	Virtual DOM.
Updates	Direct, immediate.	Batched, minimal.
performance	slower for large updates	Faster due to diffing.
Developer-Control	Manual DOM manipulation	Declare via JSX.
Efficiency	Renders everything on chang	Renders only changed parts.
Ease of use	More boilerplate	clean, component-based

Ans: to the ques NO-08.

What is Event Delegation :-

Event delegation is a technique where a sigle event listener is added to a common parent (or ancestor) instead of adding separate listeners to each child elements.

It relies on event bubbling, where events start from the target element and bubble up through the DOM hierarchy to ancestor.

Why we Event Delegation?

- Reduces the number of event listeners in memory.
- Specially useful when handling events on dynamically added elements.
- Makes the code cleaner and more maintainable.

Example Scenario:-

```
<div id = " container ">
  // Buttons
  < div>.
<button onclick = " addButton() " > AddButton </button>
```

JS

```
function addButton(){
  const btn = document.createElement('button');
  btn.textContent = 'Click Me';
  btn.ClassName = 'dynamic-btn';
  document.getElementById('container').appendChild(btn);
}
```

If we use a traditional approach we have to add a click listener every time we create a new button.

```
btn.addEventListener('click', function(){
    alert('Button clicked!!'); });
```

Instead of attaching listeners to every button, attach one listener to the parent container.

```
document.getElementById('container').addEventListener
('click', function(event){ if(event.target.classList.
contains('dynamic-btn')){
 alert('Button clicked!');
}});
```

How this works:-

- When a button is clicked, the click bubbles up to the #container.
- The listener on #container checks event.target to see if the clicked element is a .dynamic-btn
- If so, it handles the event.

Ans to the ques: No-09

Using Java Regular Expressions for input Validation:—

(Regex) → Regular Expressions in Java are powerful tools for validating input formats like emails, phone numbers, password etc.

What is Input Validation?

Input Validation is the process of checking weather the input given by a user is in correct format.

Java regex class:—

Java has two main classes in java.util.regex for working with regex:

Class	Description.
Pattern	compiles the regex string
Matcher.	Checks if the input string matches the pattern.

Explaination of the pattern:-

Part	Meaning
^	start of the string.
[A-Za-z0-9+_.-]+	one or more valide char befor @.
@	Required literal at-symbol.
[A-Za-z0-9.-]+	One or more domain charecters (gmail)
\\.	A literal dot before domain extension.
[A-Za-z]{2,6}	2 to 6 letter like .com, .org
$	End of the string.

Code:

```
import java.util.regex.*;
public class EmailValidator{
public static void main(String[] args){
String email = "example.user123@gmail.com";
String regex = "^[A-Za-z0-9+_.-]+@[A-Za-z0-9.-]+
\\.[A-Za-z]{2,6}$";
Pattern pattern = Pattern.compile(regex);
Matcher matcher = pattern.matcher(email);
```

```
if (matcher.matches()){
    sout ("Valid");}
else {sout ("Invalid");
}}}
```

Key methods:-

Pattern ~~Compile (regex)~~

class	method	purpose
Pattern	compile (regex)	Compile a regex string into a pattern.
Matcher	matcher (input)	Creates a matcher for a specific input string.
Matcher	matches()	Returns true if the input fully matches the regex.

Ans: to the guess: NO-10

What is custom annotation in Java?

Custom annotations in Java are user defined annotations that act as metadata we can attach to code elements. They do not change the code directly but when combined with reflection, they allow us to influence program behavior dynamically at runtime. They are similar to built in annotations like @Override, but we can create them ourself.

How it influences Runtime Behavior:

1. Developer adds annotation: @Runtime @RunImmediately (times = 3)
2. At runtime; the Annotation Processor reads the annotation using reflection.
3. It extracts the metadata (times = 3) and uses it to call the method 3 time.
4. This way behavior is controlled by the annotation not by hardcoded logic

Code:

```
import java.lang.reflect.Method;
public class AnnotationProcessor{
  public static void main(String[] args) throws Exception
{
  MyService service = new MyService();
  Class<?> clazz = service.getClass();
  for (Method method : clazz.getDeclaredMethods())
{if (method.isAnnotationPresent(RunImmediately.class)){
  RunImmediately ann = method.getAnnotation(RunImmediately
                                            .class);
  int times = ann.times();
  for (int i=0; i< times; i++){
    method.invoke(service);
}}}}
```

Ans: to the ques: NO – 11

The singleton design pattern is a creational design pattern that ensures a class has only one instance and provides a global point of access to it.

How singleton ensures only one instance is created:-

- Private constructor:- The constructor is declared private so that no other class can instantiate it directly.
- Static Instance:- A private static variable holds the single instance of the class.
- Public Access method:- A public static method (usually named getInstance()) returns the static instance.

Thread safety in Singleton Implementation:-

In a multithread environment, the basic implementation may result in multiple threads creating separate instances. Therefore, thread safety is crucial to ensure that only one instance is ever created, even when multiple threads try to access it simultaneously.

Thread safety singleton Implementation:-

1. Synchronized method:-

```
public class Singleton{
    private static Singleton instance;
    privat singleton(){}
    public static synchronized singleton getInstance(){
    if (instance ==null){
        instance = new singleton();
    }
    return instance;
}
}.
```

Ans: to the ques: NO-12

JDBC is an API that allows Java applications to interact with rational database such as MySQL, O JDBC.

How JDBC Manages Communication:

JDBC act as a bridge between the Java application and the database:

1. JDBC Driver: Each database has a specific JDBC driver that implements the standard JDBC interfaces.
2. Connection: The application requests a connection to the database via the driver.
3. Statement Execution:- SQL queries are sent via statement or PreparedStatement.

4. Result Retrival: The database returns the result which is fetched using Resultset.

5. Resource Management:— JDBC resources like connection, statement, and resultset are closed after use.

Steps of 'Execute a select query and Fetch Results':—

```
import . java. sql. *;
public class SelectExample{.
public static void main (String [] args ){.
String Url = " Jdbc : mysql: // localhost: 3306 / Studentdb4 ";
String user = "root";
String pass = " 12345 ";

Connection conn = null;
Statement stmt = null;
Resultset rs = null;
try { class . forname (" com. mysql . cj-jdbc. Driver ");
    conn = DriverManager. getConnection (Url, user, pass);
    stmt = conn. createStatement ();
    String query = " select id, name, source from student ";
    rs = stmt. executeQuery (query);
    While (rs. next ()) {
    int id = rs. getInt ("id");
    String name = rs. getString ("name");
    int score = rs. getInt ("score");
```

```
Soy ("ID" +id+ ", Name :" +name);
}
catch (ClassNotFoundException e)
{
sout ("Mysql JDBC Driver Not found");
e.printStackTrace();
}
}
```

Ans: to the ques: NO-13

Servlets and JSPs in MVC Architecture:-

The model view controller (MVC) architecture is widely used design pattern in web applications that separates the application login into three distinct components:

- Model : Represents the data and business logic
- View : Responsible for displaying the data.
- Controller: Manages the flow between the Model and View.

In a Java web application:

- servlet act as controllers.
- JSPs act as views.
- JavaBeans or POJOs act as model.

How Servlets and JSPs work together in MVC:

1. Client sends a request to server.
2. Servlet receiving the requests, process, input, interacts with the model and prepare data.
3. Model is used to retrieve or update data.

4. servlet sets data as request attributes and forwards the request to a Jsp.

5. Jsp reads data from the request attributes and displays the response to the user.

Use case:

Model (Java class) student.java:

```
public class student{
    private int id;
    private String name;
    private int score;
public student (int id, String name, int score){
    this.id = id;
    this.name = name;
    this.score = score;
}
Getters
}
```

Controller (StudentController.java) —

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class StudentController extends HttpServlet{
int id = Integer.parseInt(request.getParameter("id"));
Student student = new student(id, "Alice", 90);
```

```
request.setAttribute("student", student);
RequestDispather dispatcher = request.getRequestDispatcher
("studentView.jsp");
}
```

3. View (JSP) StudentView.JSP:

```
<%@ page contentTyp = "text/html; charset=UTF-8" language
= "Java" %>.
<%@ page import = "your.package.Student" %>
<% student student = (student).
%>
<html>
<head> <title> student Info </title> </head>
<body>
<h2> Student Details </h2>
<p> ID : <%= student.getId() %> </p>.
<p> Name : <%= student.getName() %> </p>.
<p> Score = <%= student.getscore() %> </p>.
</body>
</html>.
```

Ans: to the quest No-14

A Java servlet is a server-side Java program that handles HTTP requests an generates dynamic web content.

Servlet life cycle stages:-

The servlet life cycle includes the following stages:

1. Loading and Instantiation: The servlet container loads the servlet class and creates an instance.
2. Initialization (init()): called once after the servlet is instantiated to perform one-time setup.
3. Request Handling (service()): Called for every client request to process and generate responses.
4. Destruction (destroy()): Called once before the servlet is taken out of service, for cleanup

Key methods in servlet lifecycle:-

1. init (ServletConfig config)
   - Called only once when the servlet is first created.
2. service ():
   - Called every time the servlet receives a new client request.
   - It delegate to appropriate method like doGet() doPost() ...

3. destroy():
• Called only once when servlet is about to remove.

Concurrent Request Handling and thread safety:

Concurrency:
.. By default, a servlet container uses a single instance of the servlet class and creates a new thread for each client request.

Thread safety:
Because a single instance handles multiple threads, shared resources can lead to race conditions, data corruption

Ans: to the qus: No-15

In Java servlets, the servlet container uses a single instance of a servlet to serve multiple client requests concurrently.

Problem can occur:
- Race condition.
- Incorrect results.
- Unpredictable behavior.
- Security vulnerabilities

Unsafe version:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class CounterServlet extends HttpServlet{
```

```
private int counter = 0;
protected void doGet (HttpServletRequest request, HttpServl
Response response).
  throws ServletException, IOException{
  counter ++;
  response.setContentType ("text/plain");
  repons.getWriter().println ("Visitor count: "+Counta
}
```

Ans to the ques NO-16

The MVC pattern is a widely adopted architectural design pattern used to separate concerns in application development. In a Java web application, the MVC pattern divides the application into-

- Model
- View
- Controller.

Model

- Represents the data and business logic of the application.
- Interacts with the database
- Contains Java classes.

View

- Represents the presentation layer.
- Implemented using JSPs, HTML, CSS.

• Reads data passed by the controller and displays it.

Controller:
• Act as an intermediary between the model and the view.
• Implemented using servlet or controllers in frameworks like Spring MVC.
• Handle user input, calls model methods and forwards results to the view.

Advantage of MVC:

Maintainability:
• Clear separation makes it easier to update or debug individual components
• UI designers can modify JSPs without touching.

Scalability:
• New feature can be added easily.
• Each layer can be scaled independently.

Reusability:
• Business logic in the model can be reused.
• Same data can be displayed in multiple view.

Ans to the ques NO-17

In a Java EE web application, the servlet acts as the Controller in the MVC pattern. It handles incoming client requests, interacts with the Model and forwards data to the view.

Flow overview:

1. Client sends a request.
2. The servlet (Controller) processes the request parameters.
3. The Servlet calls the Model to handle business logic
4. The servlet sets data as attributes in the HttpServletRequest
5. The servlet forwards the request to a JSP.
6. The JSP accesses the attribute and renders the response

Displaying user details.

```
Import java.io.*;
Import javax.servlet.*;
Import javax.servlet.http.*;
public class userServlet Extends HttpServlet{
```

protected void doPost(HTTPServletRequest request, HTTPServletResponse response)
throws ServletException, IOException{
String name = request.getParameter("name");
String email = request.getParameter("email");
User user = new User(name, email);
request.setAttribute("user", user).
}

Ans to the ques: NO-18

To maintain session data across multiple client request, Java servlets provide various session tracking techniques

1. Cookies.
2. URL Rewriting
3. HttpSession.

1. Cookies:

How it works:

- Cookies are small pieces of data stored on the clients browser
- A servlet sets a cookie using the HttpServletResponse.addCookie
- The client sends the cookie back with each subsequent request.

Advantages:

- Simple and widely used.
- Data persists even after the browser is closed.

URL Rewriting:

How it Works:

- Session information is appended to the URL as a query parameter.
- The server extracts data from the URL on each request.

Advantages:

- Works even if cookies are disabled
- Requires no browser support

HTTP session:

How it Works:

- HTTP session stores session data on the server.
- Each client gets a unique session ID.
- You can store any type of object in the session.

Advantage:

- Stores any Java objects, not just strings.
- More secure, as data is stored server-side.
- Simplifies session management.

Ans: to the ques NO-19

In Web application, HTTP is stateless, meaning the server cannot remember previous client requests.

How HttpSession Works Across Multiple Requests:-

User logs In:

- After successful authentication, the server creates or retrieves an existing HttpSession object.
- The user's login information is stored in the session object.

Session ID creation:

- The server generates a unique session ID.
- This ID is usually sent to the client in a cookie.
- The client sends this cookie back on each request, allowing the server to identify the session.

Subsequent Requests:

- The client includes the session ID in every request.
- The server uses this ID to retrive the associated session object.
- The user remains "logged in" as long as the session exists.

Handling Session Timeout and Invalidation:

1. Session Timeout:

Default timeout usually 30 min.

Manual invalidation:

- Sensitive data is cleared from memory
- The session ID become invalid
- Prevents session fixation and unauthorized access after logout

Ans: to the ques: NO -20

Request Handling in Spring MVC:

When a user submits a request from the browser, spring MVC process it in the following way:

Spring MVC lifecycle:

1. User sends HTTP request.
2. The DispatcherServlet receives the request and act as the front controller.
3. It looks for the appropriate method in class ~~or~~ annotated with ~~@Controller~~ @Controller using @RequestMapping.
4. The controller method is executed
5. A Model object is populated with data.
6. The controller returns a view name.

7. The view is rendered with the data from the Model and sent back to the browser.

Key annotations:

@ controller:
1. Marks a class as a web controller.
2. Allows spring to detect and register it as a component.

@ request Mapping / @ post mapping.
1. Maps URLs to methods.
2. Can define path, HTTP method, parameters.

Ans to the ques NO-21.

In spring MVC, the DispatcherServlet is the central component that manges and controls the entire request handling prosess.

Request Procesing Workflow with DispatcherServlet:

When a browser sends an HTTP request to a spring MVC web application, the following steps occur.

1. Request Interception by DispatcherServlet:

- The DispatcherServlet receives the HTTP request
- It is configured in web.xml or via annotation like @ springboot Application.

2. Handler Mapping:

- The Servlet consults a HandlerMapping to find the controller.
- It matches the request URL and HTTP method with controller mapping like @ RequestMapping.

3. Handler Execution:

- Once the appropriate controller in found, the Dispatcher servlet invokes the method
- The controller process the request, possible interact with a service.

4. View Resolution:

- The returned view name is passed to the view resolver.
- The view resolver maps the logical name to an actual Jsp.

Ans: to the ques No-22

In JDBC both statement and PreparedStatement are used to execute SQL queries. However, PreparedStatement provides significant advantages over statement in terms of performance.

Advantage of prepared statement:

1. Security: Prevents SQL injection:

- Prepared statement separate SQL code from user input
- It uses placeholders (?) and bind values safely, preventing malicious injections.

2. Performance:

- Prepared statement queries are precompiled by the database
- The SQL execution plan is reused for each execution, improving performance in loops or repeated queries.

3. Code readability:

- Clear separation between SQL and Java code.
- Easier to update and debug.

Ans to the ques NO-23

In JDBC a resultset is an object that stores the result of executing a sql query. It act like a cursor.

How Resultset Works

- We execute a query using statement
- The query Returns a Resultset object
- We use the next() method to move through rows one by one.
- Use getString(), getInt(), and similar methods to retrieve column values from the current row

Key method of resultset.

next() - Moves the cursor to the next row.
getString() - Retrieves a String from the specified column
getInt() - Retrieves an int from the specified column.

Explanation of Methods.

- rs.next(): Moves the cursor to the next row in the resultset.
- rs.getInt(): Gets the int value from the 1st column

of the current row.

- rs.getString : Gets the string value from the name column
- rs.getStrs. . Gets the string values from the array.

Ans to the ques NO-24

JPA (Java Persistence API) is a specification in Java EE that provides a standard way to map Java objects to relational database tables.

Key Annotations in JPA:

@Entity — Marks a Java class as a JPA entity.
@Id — Specifies the primary key of the entity.
@GeneratedValue — Specifies that the primary key value should be auto-generated.

Example:

```
import javax.persistence.*;
public class Student{
```

```
private int id;
private string name;
"       "  email;
public student() {}
public student (string name, string email)
{ this.name = name;
  this.email = email;
}
// getters setter
}
```

Ans to the ques NO-25

In Java Persistence API is the primary interface used for interacting with the persistence context. It provides several methods to create, update, and delete records in a database.

The most commonly used methods are-

1. **Persist:**

Purpose:
- used to insert a new record into the database
- Makes a new entity instance managed and persistent.

Behavior:

. After calling persist(), the entity becomes managed and will be stored in the database
. Only works with ~~e~~ new entities.

2. Merge (object):

Purpose:

. Used to update an existing record or to reattach a detached object to the persistence context.

Behavior:

. If the entity already exists in the database, its state is updated.
. If the entity is detached or not managed, a new managed instance is created and returned.
. Does not change the original object; returns a new managed copy

3. Remove (Object entity):

Purpose:

. Used to delete an entity from the database.

Behavior:

. The entity must be managed.

• If not, you must first fetch or merge it then call remove().

Ans to the quest NO-26

1. Student Entity:

```
import jakarta.persistence.*;
public class student{
private int id;
private string name;
private string email;
public student()<>
public student (string name, string email){
this.name = name;
this.email = email;
}
// Getters setters.
```

2. StudentService class:

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.service.
import java.util.List;
import java.util.Optional
```

```
Public class studentservice{
 private student addstudent (student student){
    return repo.save (student);
   }
public List<student> getAllstudents(){
  return repo.findAll();
}
public student getstudentById (int id)
 {return repo.findById.orElse(null);
 }
public void deletestudent (int id){
  repo.deleteById(id)
}}
```

Ans to the ques NO-27.

To learn how Spring Boot simplifies the development of Restful APIs and implement a Rest controller that handles JSON data using @RestController.

Create the student Class:

```
public class Student{
private int id;
private string name;
private string email;
public student(){}
public student(int id, string name, string email)
{this.id = id;
this.name = name;
this.email = email;
}
//Getters setter.
}
```

Create a Rest Controller:

```
import org.springframework.web.bind.annotation.*;
import java.util.*;
public class StudentController{
 private List<Student> studentList = new ArrayList<>();
 public List<Student> getAllStudents(){
 return studentList;
 }
 public String addStudent(Student student){
  studentList.add(student);
  return "Student added sucessfully.";
 }
}
```

Ans: to the the quest NO-28

Feature	@Controller	@restcontroller.
Purpose	Used to return views in web applications	Used to return data for Rest API.
Returns	Views templates	Data object directly written to response body
Annotation Behaviour.	Requires @ResponseBody on each method to return JSON	Combines @controller + @ Responsebody so no need for extra annotation
Use case	Traditional MVC apps	Restful APIs.

Example:

Using @Controller:

```
@Controller.
public class WebController{
  public String homePage(){
    return "home";
  }
}
```

Using @RestController:

```
@RestController
public class ApiController{
  public String getMessage(){
    return "Hello from REST API";
  }
}
```

Rest API Design for Library System:

Resource: Book

Each book has properties like:

- id (int)
- title (String)
- author (String)
- isbn (String)
- available (boolean).

Ans: to the ques: No-29

Maven:

Maven is a powerful build automation and project management tool used in Java projects. It handles

- Dependency Management
- Project Compilation,
- Packaging and testing.
- Running Application.
- Life cycle management

How maven Works:

1. **Dependency Management:**

- All required libraries are declared in the pom.xml.
- Maven download those libraries from the Maven Central Repository or Custom repositories and places them in the project classpath.
- Version conflicts are resolved automatically based on Spring Boot dependency management system.

2. Build lifecycle:

Maven follows a standard build lifecycle with several phases:

phase	Description
Validate	Validate the project structure.
Compile	Compiles source code.
test	Runs unit tests.
package	Packages the compiled code into JAR.
verify	Checks integration.
Install	Installs the built package into the local repository.

Ans: to the quesNO-30

In springboot Maven and gradle are too most popular build tools. Both are used to

- Manage dependencies.
- Compile source code
- Run test.
- Package and deploy application.

Comaparison.

Feature	Maven	Gradle
Language	XML (pom.xml)	Groovy / Kotlin DSL
Vebosity	More verbose dut to XML	More concious and readable
performance	Slower	Faster.
Dependency Handling	Centralized via <dependency> in XML	uses implementation, test Implementation.
Custom Build logic	Harder to customize	Easy

Modular arithmetic

Tooling support	Strong IDE support	Strong support by intelliJ idea.
learning curve	Easier for Beginner	Slightly steeper due to DSL

These are the comparision between Gradle and Maven