

IT22034.

Rahat Bin Hafiz

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 subclasses, even if those subclasses are in different packages.

• Child in the same package:-

Parent.java :-

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

Child.java :-

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

QUESTION

Static and Local

Main.java

package mypackage;

public class Main

public static void main (String [] args) {

Child child = new Child();

child.showMessage();

Different package -

Parent.java:

package parentpkg;

public class Parent

protected String = "Love You";

>

Child.java:

package childpkg;

import parentpkg.Parent;

public class Child extends Parent

public void showC()

> sout (message); /Accessible

public void test()

< parent p = new Parent(); /not Accessible;

Main . Java :-

```
package childpig;  
public class Main{  
    public static void main (String [] args)
```

```
        < child c = new child;
```

```
        c.show();
```

```
>
```

```
    < () button b1ov > A 33333333
```

```
    < () button b1ov > B 33333333
```

```
    . < -> C A. etremly 3 3333
```

(button signs), park toothless and glass and park A

```
    < () button b1ov > D A park toothless
```

```
    . < -> D A park toothless
```

— park toothless in D

park, east to 2nd floor signless been 3333

and glass toothless and toothless signless been 3333

each

② Compare abstract classes and interfaces in terms of multiple inheritance. When would you use them?

Ans:

Interface:

A class can implement multiple interfaces. Java does not support multiple inheritance but allow multiple inheritance 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 inheritance)

Ex:-
abstract class Abc { abstract void method(); }
class B extends Abc { ... }.

Use an interface when:-

- We need multiple inheritance of type. A class can implement multiple interfaces but extend only one class.

When to use an abstract class:

• We can use it to share common code among related classes. Abstract classes can have concrete methods with default behavior.

• Abstract can have constructor but interface cannot.

• Features staying between sub-class and super-class

• Inheritance with abstract base class vs concrete

• Subclasses inherit both methods and fields.

• Inheritance with concrete base class
• Subclasses inherit methods and fields

• Inheritance with abstract base class

(Inherited from the base class with final methods)

• Inheritance with concrete base class

• Subclasses inherit methods and fields

③

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

Data security:-

- Variables 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 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 accountNumber) {
        if (accountNumber == null || accountNumber.trim().isEmpty())
            < sout ("Invalid account Number."); return;
        > this.accountNumber = accountNumber;
    }
    public void setInitialBalance (double bal) {
        if (bal < 0)
            sout ("Initial Balance cannot be negative.");
        return;
        > this.balance = bal;
    }
    public String getAccountNumber () { return AccountNumber; }
    public double getBalance () { return balance; }
}
```

```
public class BankTest {
    public static void main(String[] args) {
        BankAccount acc = new BankAccount();
        acc.setAccountNumber("ABC12345");
        acc.setInitialBalance(5000);
        System.out.println("Account " + acc.getAccountNumber());
    }
}
```

~~Ans to -~~ Ans: to the ques! No-04

Q) Find the kth smallest element.

```
import java.util.*;  
public class 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);  
            System.out.println (k + "th element is " + list.get (k-1));  
        }  
    }  
}
```

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";  
        TreeMap <String, Integer> map = new TreeMap <>();  
        String [] words = text.toLowerCase().split ("\\W+");  
        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 ());  
>
```

```
>>>
```

⑤ 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 Registration parking 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)
            < Registration car = pool.getNextCar();
            if (car == 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 = new parkingPool();
```

```
        parkingAgent agent1 = new parkingAgent (1, pool);
```

```
        parkingAgent agent2 = new parkingAgent (2, pool);
```

```
        agent1.start();
```

```
        agent2.start();
```

```
        String [] carNumber = {"ABC 1234", "King 1234"};  
        for (String carNm : carNumbers)
```

```
            new Thread (c) {
```

```
                RegisterParking request = new RegisterParking  
(carNm);  
                pool.requestParking (request);  
            }.start();
```

try < do - one sleep with other

Thread.sleep(200);
out.println("and");

Catch (Interrupted Exception e)

Thread.currentThread().interrupt();

 **Y**ou must be fast to keep up with
the world as it is changing rapidly. But
you can't do that unless you have
the right tools.

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 library 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
User case	Small to medium XMLs, heading updates	Large XMLs, read-only, streaming.

Example of SAX over DOM! —

so Imagine you we are building a log monitoring tool that scans a massive 2GB 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 we can trigger events when those tags are encountered.
- It 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.

- Only those changes are applied to the real Dom.

Differing Algorithm (Reconciliation):-

React uses a highly efficient differencing algorithm based on the following assumptions:

- Two elements of different types will produce different trees.
- 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 DOM tree.
- Comparisons:
 - 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 creates a new VDom for <p> 1 </p>.
2. Compares it to the previous VDom ' <p> 0 </p>' .
3. Detects a change only in the text node inside <p>
4. Updates only that node in the real Dom, instead of re-rending the whole <div> .

VDom VS DOM

Feature	Traditional DOM	Virtual DOM
Update	Direct, immediate.	Batched, minimal.
Performance	slower for large updates	Faster due to diffing.
Developer Control	Manual DOM manipulation	Double via JSX.
Efficiency	renders everything on change	renders only changed parts.
Code of Use	More boilerplate code.	clean, component-based.

Ans to the ques No - 08,

What is Event Delegation:-

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

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

Why use 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()"> Add Button </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 ~~bit~~ 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 one. If so, it handles the event.

Ans to the ques No - 09.

Using Java Regular Expression for input Validation:-

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

What \Rightarrow Input Validation?

Input Validation is the process of checking whether 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 compiler the regex string
Matcher checks if the input string matches the pattern.

Example: $\text{String str} = \text{Pattern.compile("^\w{5}-\w{4}$")};$

$\text{if(str.matcher("12345-6789").matches())}$

Output: true. means = pattern matching
(true) which means = valid string

Explanation of the pattern:-

Part

Meaning

[A-Za-z0-9_].+ start of the string.

[A-Za-z0-9_].+ one or more valid char before @. required literal '@' symbol.

[A-Za-z0-9_.]+ One or more domain characters

with option to add @ (gmail) literal dot before domain extension.

[A-Za-z0-9_].+ 2 to 6 letter like .com .org
@ End of the string within

Code:

```
import java.util.regex.*;
public class EmailValidator{
    public static void main (String[] args){
        String email = "example.user@gmail.com";
        String regex = "[A-Za-z0-9_].*[A-Za-z0-9_]*@[A-Za-z0-9_].*[2-5].*\$";
        Pattern pattern = Pattern.compile (regex);
        Matcher matcher = pattern.matcher (email);
```

```
if (matcher.matches()) {
```

```
    System.out.println("Valid");
```

```
else { System.out.println("Invalid"); }
```

→ It prints the output based on the result of the matches method.

Key methods -

Pattern → compile(Regex)

Pattern → match(Regex)

Class → compile(Regex) → Pattern → 'Purpose'

Pattern → matcher(Regex)

Matcher

Matcher → matcher(Regex)

Matcher

Matcher → matcher(Input)

Matcher

Matcher → matcher(C)

Matcher

Matcher → matcher(Regex)

Matcher

Matcher → matcher(Regex)

Matcher

Matcher → matcher(Regex)

Matcher

Matcher → matcher(Regex)

Matcher

Ans: To the question No-10

What is custom annotation in Java?

Custom annotations in Java are user-defined annotations that act as metadata you 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 ourselves. How it influences runtime Behavior:-

1. Developers adds annotation: `@Runtime @RunTime directly (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 times.
4. This ~~way~~ behavior is controlled by the annotation not by hand coded 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(RunImmedately.class))<
    RunImmediately annotation = method.getAnnotation(RunImmedately.class);
```

```
int times = annotation.times();
```

```
for (int i = 0; i < times; i++)<
```

```
    method.invoke(service);
```

```
<
```

```
<
```

```
<
```

Annotations are used to add functionality to the code without changing the code directly. Annotations are like metadata or annotations.

Ans: to the ques: NO - 11

The ~~size~~ 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 multithreaded 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;
    private 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 application to interact with relational database such as MySQL, A ~~DB~~.

How JDBC Manages Communication:

JDBC acts 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 interface.
2. Connection: The application requests a connection to the database via the driver.
3. Statement Execution: - SQL queries are sent via statement or Prepared Statement.

4. Result Retrieval: 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/studentdb";
        String user = "root";
        String pass = "12345";
        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;
        try {
            Class.forName ("com.mysql.jdbc.Driver");
            conn = DriverManager.getConnection (url, user, pass);
            stmt = conn.createStatement ();
            String query = "select id, name, score from student";
            rs = stmt.executeQuery (query);
            while (rs.next ()) {
                int id = rs.getInt ("id");
                String name = rs.getString ("name");
                int score = rs.getInt ("score");
            }
        } catch (Exception e) {
            e.printStackTrace ();
        }
    }
}
```

```
System.out.println("ID: " + id + ", Name: " + name);
    } catch (ClassNotFoundException e) {
        System.out.println("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 application that separates the application logic into three distinct components:

- Model : represents the data and business logic.
- View : responsible for displaying the data.
- Controller : Manager the flow between the Model and View.

In a Java web application:

- Servlet act as controllers.
- JSPs act as views.
- Java Bean or POJOs act as model.

How servlets and JSPs work together in MVC.

1. Client sends a request to server.
2. server receives the requests, processes, input, interacts with the model and preparing data.
3. Model is used to retrieve or update data.

4. servlet sets data as request attributes and forward the request to <.jsp.

5. JSP reads data from the request attributes and display 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, "Alvey");
}
```

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

```
>
```

3. View (JSP) StudentView.jsp:

```
<%@ page contentType="text/html; charset=UTF-8"%>  
= <%@ page %>.  
<%@ 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>.
```

Amt. to the quest No-19

A Java servlet is a server-side Java program that handles HTTP requests and 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 delegates to appropriate method like doGet() or doPost()

3. `destroy()`:
- Called only once when servlet is about to remove.
 - Correct request handling and thread safety.

concerns:

- 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 condition, data corruption.

Ans: To the Qn: Notes

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 {
    public void service(HttpServletRequest req, HttpServletResponse res) {
        res.setContentType("text/html");
        PrintWriter pw = res.getWriter();
        int count = 0;
        while (true) {
            count++;
            pw.println("Count: " + count);
        }
    }
}
```

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



Ans to the ques No-16

The MVC pattern is a widely adopted architecture/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:

• acts as an intermediaries between the model and the view.

• Implemented using servlet or controllers in framework like Spring MVC.

• Handle user input, calls model methods and forward results to the view.

Advantage of MVC:

Maintainability:

- clear separation makes it easier to update or debug individual components
- UI designer 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.

Flow to the client

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 HttpSession.
5. The servlet forwards the request to a JSP.
6. The JSP accesses the attribute and renders the response.

Displaying user details.

```
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;

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,

JSPs, 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 client's 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 in a query parameter.
- The server extracts data from the URL through query parameters.

Advantages

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

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 quest 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 retrieve 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 is usually 30 min.

Manual invalidation

- " sensitive data is cleared from memory
- The session ID become invalid
- Prevent session fixation and change the user session after logout

Ans to the ques No 20

Request Handling in Spring MVC:

When a user submits a request from the browser, spring mvc processes 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 annotated with @Controller annotation.

Request Mapping:

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 we the DispatcherServlet is the central component that manages and controls the entire request handling process.

Request Processing Workflow with Dispatcher Servlet-

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 is found, the Dispatcher sends it invoke the method
- The controller processes 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 quest No-12

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

Advantage of prepared statement:

1. Security: Prevents SQL injection:

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

2. Performance:

- Prepared statement queries are precomputed 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 acts 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 7th column

of the current row.

- rs.getString : Gets the string value from the name column.
- rs.getString : Gets the string value from the email.

Ans: to the query 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:

- ① Entity — Marks a Java class as a JPA entity.
- ② Id — Specifies the primary key to the entity.
- ③ GeneratedValue — Specifies that the primary key value should be auto-generated.

Example

import javax.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

Object manipulation methods

Ans to the quest 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 method 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 new entities.

2. Merge (Object):

Purpose:

- Used to update an existing record or to refresh 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 it is managed. Its instance is created and returned.
- Does not change the original object; return a new managed copy.

3. Remove (Object entity):

Purpose:

- Used to delete an entity from the database.

Behavior:

- The entity may 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.entity
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;
    }
}
```

11 Getters setters.

2. Student Service 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(id).orElse(null);
    }
    public void deleteStudent(int id) {
        repo.deleteById(id);
    }
}
```

Ans to the que 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 successfully.";  
    }  
}
```

Ans to the question NO-28

Feature	@controller	@restcontroller
Purpose	Used to return Views in Web applications	Used to return data for rest API
Returns	Views template	Data object directly written to response body
Annotation Behavior	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

Resources: Books

Each book has properties like:

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

Books (entity/resource) no collection function

Collection function for books off as well ready

Published books no diff func ready

Booked books func nothing no book

... missing

Ans: to the quest 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 place them in the project classpath.

Version conflicts are resolved automatically based on Spring Boot's dependency management system.

2. Build Lifecycle:

Maven follows a standard build lifecycle with several phases:

<u>phase</u>	<u>Description</u>
validate	Validates 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 question

- In spring boot Maven and Gradle are two most popular build tools. Both are used to
 - Manage dependencies.
 - Compile source code.
 - Run test.
 - Packaging and deploy application.

Comparison

Feature	Maven	Gradle
Language	XML (pom.xml)	Groovy / Kotlin DSL
Verbosity	More verbose due to XML	More concise and readable
Performance	Slower	Faster
Dependency Handling	Centralized via dependency in XML	was implementation, fast implementation.
Custom build logic	Harder to customize	Easier

Modular arithmetic.

Tooling support	Strong DDF support	Strong support by Intel's idea.
Learning curve	Easier for Beginner	Stable & known due to DSL

Thus are the comparison between Grindle and Myria.