2 Jackson (for JSON Serialization)

→1) QuizController. java

Type: Controller

Purpose: Handles REST API endpoints.

Methods

getQuestions() → Get /api/questions

2. Question Java

Type : model

Purpose: Represents a single Quiz Question

method: getID(), getQuestion (), getOption()

Classes

QuizController → handle REST API

Question — represent single ques

AnswerRequest — carries the user answer

@ RestController

API request নেবে আর JSON আকারে রেসপন্স পাঠাবে।

@ RequestMapping("/api")

এই annotation বলে দেয় যে, এই controller এর সব এন্ডপয়েন্ট /api দিয়ে শুরু । যেমন /api/questions, /api/submit.

Summary

1. Spring Boot Starter Web

- Used for building REST APIs

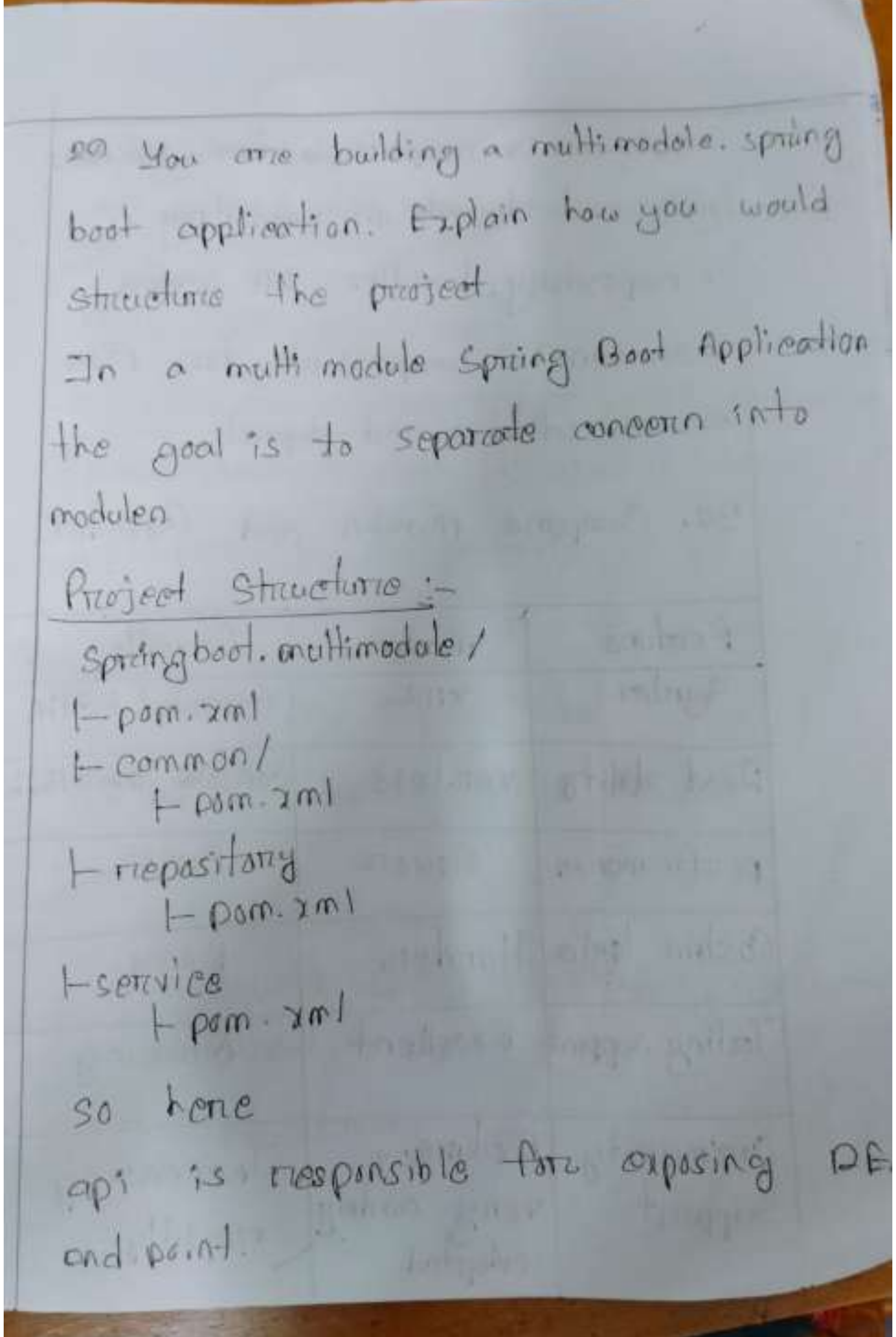
Provides:

- @ Rest Controller
- @ RequestMapping, @postMapping, @ Get Mapping
- @ ResponseEntity.

· service is responsible for business logic and depends on repository.
· repository handles DB access.
· common is responsible for DTO's enums, utilities and depends .

30. Compare maven and Gradel.

Feature	maven	Gradle
Syntax	XML	Groovy/ Kotlin
Read ability	verbose	console, more readable
performance	slower	faster
Custom logic	Harder	Easy
Tooling support	Excellent	Growing
Community Support	mature, very widely adopted	Increasing rapidly.



20 You are building a multimodule. spring boot application. Explain how you would structure the project

In a multi module Spring Boot Application the goal is to separate concern into modules

<u>Project Structure :-</u>

```
springboot.multimodule/
|- pom.xml
|- common/
    |- pom.xml
|- repasitory
    |- pom.xml
|- service
    |- pom.xml
```

So here

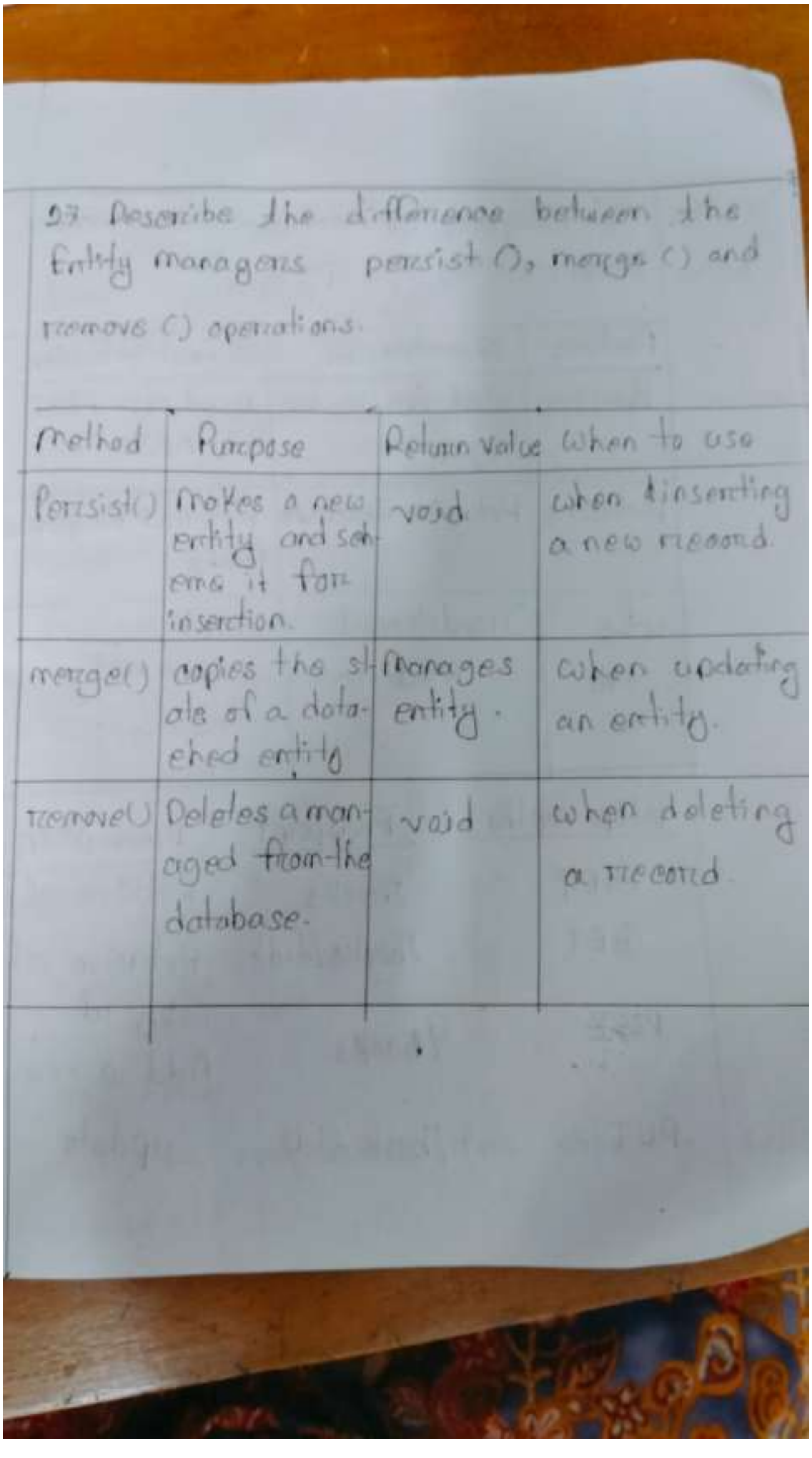
api is responsible for exposing DE

end point

28. Compare @RestControllerr and @controllerr in spring Boot

Featurre	@controllerr	@RestControllerr
Purrpose.	used forr renderr-ing views.	Used forr REST ful API's
Returrns	Returns view name	Returns Response Body
use case	Traditional mvc	web services

Http method	Endpoint	Description
GET	/books	Retrive all books
GET	/books/{id}	Retrrive a book by id.
POST	/books	Add a new
PUT	/book{id}	update.



27. Describe the difference between the Entity managers persist (), merge () and remove () operations.

Method	Purpose	Return value	When to use
Persist()	makes a new entity and scheme it for insertion.	void	when inserting a new record.
merge()	copies the state of a detached entity	Manages entity.	when updating an entity.
remove()	Deletes a managed from the database.	void	when deleting a record.

{

26. How does Preparedstatement improve performance and security over statement in JDBC ?

(i) Security :

(ii) Performance

(iii) example.

public class string {

public static void main (String[] args) {

String url = "jdbc : mysql ://localhost 3306

String insert SQL = "insert into student values (?, ?);

try (Connection conn = DriverManager.getConnection url, username , password);

pstmt . setString (1, " Jmroe ");

}

}

3. Repository Interfaces

import org.springframework.data.jpa.repository.

. . . .

. . . .

26. How does ~~no~~ Boot simplify the development of RESTful service ?

Spring boot greatly simplifies the development of RESTFUL services :—

i) Auto configuration

ii) Embeded server

iii) Spring web controllers

iv) Reduced Boilerplate

✧

24. Design a simple CRUD applications using spring and mySQL to manage student records.

Project OverView:

i) Add as student.
ii) Get all or single students.
iii) Update a student data.
iv) Delete a student.

1. Database Table

```
create database studentdb;
use studentdb;
- - - - - - - - - - -
- - - - - - - - - - -
```

2. Entity class -

student. java

```
import jakarta.persistence;
@ Entity
```

23. How does JPA manage the mapping between Java objects relational tables ?

Entity defination:

• Java classes are marked as JPA entities using the @Entity annotation.

Field to Column mapping:

• By default , JPA maps fields inan entity by class to column in the corresponding database.

Relational mapping:

JPA provides annotation to manage relation ships between entities, which corries ponds to relation.

22. What is a Resultset in JDBC and how it is used to retrun data from a my SQL database ?

In JDBC, a ResultSet is an object that holds the data returned from executing a select query on a relational data

How it retrives data:

- Establish a database connection
- Create a statement.
- Execute the query
- Iterate through the ResultSet.
- Retrive column values
- Close Resources.

1. Receive Request:

All request mapped to spring mvc are first received by dispatcher servlet.

2. Delegates to Handle mapping:

. The servlet consults Handlemapping beans to determine which controller method should handle

3. Invokes the controller:

. The identifiers controllers method is invoked with any parameters

4. Processes Return value:

. The controller' method returns a logic view data.

Examples of login-form submission:

```
@ controller
public class LoginController {
 @ postMapping ("/login")
public String handles login(
@ Request String username;
@ Request String password.
```

21. Spring mvc uses the Dispatcher servlet as a front controller. Describe it's role in the request processing?

Dispatcher servlet Interaction Diagram:

$$\begin{tikzcd}
\text{Browser Request} \arrow[d] \\
\text{DispatcherServlet} \arrow[d] \\
\text{HandleMapping} \arrow[d] \\
\text{Model} \arrow[r] & \text{View Name}
\end{tikzcd}$$

4. It looks for a @controller method, mapped to the request URL.

5. The matched @ controller method process the request.

The role of annotations in separating business

1. @ Controller →

- Keeps the business logic in service.
- Connects the request routing to backend processing.

2. @ RequestMapping →

- Directs with controller method separates both besiness.

3. Model Object →

- Avoids onbedding data processing logi in views.

Invalidation

1. You can explicity invalidate a session using → request.getSession().invalidate();

20. Explain how spring mvc handles the Http request from a browser. Describe the role of @ controller, @ requestMapping and model object in separating business logic from program action.

Spring mvc Request Handling flow:

when a browser sends a request -

1) The Dispatcher Servlet receives the request.

· This ID is usually sent to the browser via a cookie.

ii) Subsequence Request:

· The browser automatically sends the JS
ESSION ID cookie.

· The server uses this ID to retrieve the existing session object.

Session Time Out and Invalidation

1. A session will expire automatically after a part of inactivity.

2. This is configured in web.xml.

3. After the time out, the session objec
is destroy and the session ID becom
valid.

Limitations:

- Security risk
- Bookmarked URL is leaked session info.

Ideal use case:

Public web apps where cookies can't be used

19. A web application stores users login info using Httpsession. Explain how the session works across multiple request and session works.

i) User login In

- A new session is created.
- The server generated a unique session ID.

Cookies: . small piece of data stored on the client side

. Sent with each HttP request..

. Used to track user info.

Advantages:

- persistent across sessions.
- Easy to implement.

Limitations:

- Limited insize.
- Security risks.

Ideal use case:

- Remembering user preferences.

URL rewritting:

. session ID is manually added to every URL as parameter.

<body>
<h2> student details </h2>
<p> Name: <%= student.getName() %> </p>
<p> Age: <%= student.getAge() %> </p>

19. Compare and contrast cookies, URL rewritting and Httpsession as method for session tracking in servlet.

Cookies	URL rewritting	HttpSession
store session id or data in browser cookies	Appends session id to the URL as a parameter	Stores data on server with session ID in client.
storage Location client.	client	server.
persistence beyond browser	only active during session.	Exists for session duratio
Security medium	Low security	High security.

Example: student info

1. <u>model class</u>:-

```
public class Student {
    private String name;
    private int age;
private Student(string name, int age){
  this.name = name;
    this.age = age;
public string getName(); { return name;}
public int getage (){
                    return age;}
```

2. <u>JSP (view)</u>

```
<%@ page import = "your.package.student
<% student student = (student)request.getAtt
(" student Data");
%>
<html>
<head> <title> student Info </title </head
```

17. In Java. EE application, how does a servlet controller manage the flow between the model and the view? Provides a brief example that demostrates data?

In a java EE application, a servlet controller manages the flow between the model and view by:—

1) Receiving request from the clint
2) Calling model classes to process on data fetch from the database.
3) Setting attributes on the request scope.

Flow overview:

client → servlet → Model → JSP (view) → clie

17. In Java. EE application, how does a servlet controller manage the flow between the model and the view? Provides a brief example that demostrate data?

In a java EE application, a servlet controller manages the flow between the model and view by:—

1) Receiving request from the clint
2) Calling model classes to process on data fetch from the database.
3) Setting attributes on the request scope.

Flow overview:

client → servlet → Model → JSP (view) → clien

ii) examples: JSP, thymeleaf etc.

iii) Concern separate: UI rendering is kept separate from data and control logic

3) Controllers:

i) Responsibility: Acts as an intermediary between view and model.

ii) Example: Spring @ controllers classes.

iii) Concern separate: Input processing and routing are separated from data.

The mvc pattern is a widely used design pattern in Java web applications, especially withspring mvc, struts and JSF etc frameworks. It separates concern by dividing the application into

1) Model → Business logic and data

i) Responsibility: manages the application data, rules & logic

ii) Examples : Java classes that represents data, service class and DAO.

iii) Concern separated : Business logic is isolated from UI and request handling.

11) View :

i) Responsibity: Displays data to the use and handles user interfaces.

An example problem using race condition its solution using synchronization is given below:-

Example:

```
private int counter = 0;
counter ++;
response.getWriter(). println("Request member"
, + counter);
```

In this problem 2 threads run nearly a
the

16. Describe how the mvc pattern sepa
rates concern's in a Java web applicati
Explain the advantages of this structure
terms of maintainability and scalibility u
a student registration system as an
example.

when multiple threads access shared resources then the given problem arises.
In a servlet one instance serves many request simultaneously. The servlet container spawns a new thread for each request.
If instance variables are read or modified by these threads without coordination, problem arises ~~t~~ such as:-

i) Race Condition → threads interface with each other unpredictably.

ii) Data inconsistency → variables can have unexpected values.

iii) Incorrect results → one user's data may leak into another user's response.

Thread Safety concern

Since multiple threads are the same servlet instance, any shared any are mutable instance variables can be accessed by multiple threads concurrently

This can lead to -

- Race conditions
- Inconsistance results.
- Unexpected behaviour

15. A single instance of a servlet handles multiple request using threads. What prob lem can occur it's hared resources ar accessed by multiple thread ? Illustra your answer with an example and Suggest asolution using synchronization

How servlet handles Request is given below and also why thread safety matter :—

Servlet Request Handling

- The servlet container (tomcat) creates one instance of each servlet class
- For every incoming request the container spwans a new thread and invokes the servlet service method.
- This means many thread can be executing the servlet at the same time.

init ()

→ called once after the servlet is instantiated

→ used to initialized resource.

→ The servletconfig object provides access to initialization.

service ()

→ called every time the servlet receive a request.

→ It disapatches to doGet(), doPost(), etc depending on request method.

destroy ()

→ called once before the servlet is taken out of service.

→ clean up resource like closing connections

```
request.setAttribute("student", student);
}
}
```

14. Explain the life cycle of a Java Servlet. What are the roles of the init(), service() and destroy() methods? Discuss how servlet handles concurrent request and how thread safety issue may arise

The life cycle of a java servlet is managed by the servlet container and includes 3 main stages.

1. Loading and instantion

The servlet class is loaded into memor by the servlet container when the servl model: Student.Java

```
public class Student {
private String name;
private int age;
public Student (String name, int age){
this.name =name;
this.age= age;
public student (String name, int age){
}
```

Controller: Student Servlet Java

```
public class StudentServlet extends HttpServlet{
protected void doGet(HttpletRequest request,
HttpServlet)
  String studentId = request.getParameter("id");
Student student =new Student(studentId,"Imr
20);
```

13. How do Servlets and JSPs work together in a web application following the MVC (Model-View-Controller) architecture ? Provide a brief use case showing the servlet as a controller, JSP as a view and a Java class as the model.

Servlet and JSP in MVC Architecture

- Model : Handles business logic and data
- View : Handles presentation
- Controller : Handles request and responses.

How do they work:

1. User sends a request
2. Servlet (Controller) receives the request

~~fee~~ 3. The Servlet forwards the request to a JSP

Basic Singleton implementation

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

11. Discuss the singleton design pattern in Java. What problem does it solve, and how does it ensure only one instance of a class is created? Extend your answer to explain how thread safety can be achieved in a Singleton implementation.

The Singleton pattern is a creational design pattern that ensures a class has only one instance throughout the application the provides a global point of access to that instance.

It solves
i) Database connection
ii) configuration Settings
iii) Logging.

@ Retention(RUNTIME) is Key ~~for~~ ~~rf~~ reflection to access it during runtime.

<u>Use the annotation in class</u>

```
public class myTasks {
     @RunMe("Hello Task 1")
    public void task1() {
    System.out.println("Task 1 executed.");
}
 @RunMe
 public void task2() {
        System.out.println("Task 2 executed.");
}
 public void task 3() {
  System.out.println("Task 3 not annotated.")

 }
}
```

10. What are custom annotation in Java, and how can they be used to influence program behaviours at runtime using reflection ?

Custom annotation in Java are user-defined annotations that act like metadata. You can create them to mark classes, methods, fields etc.

Define a custom annotation

```
import java.lang.annotation.*;
@Retention (RetentionPolicy.RUNTIME) // Available at runtime.

public @interface RunMe {
   String value () default "Running method";
}
```

• $ - End of String

Java Code Example:

```
import java.util.regex.*;

public class EmailValidator {
 public static void main(String[] args) {
String email = "example123@gmal.com";
String regex = "^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]
                    +\\.[a-zA-Z]{2,}$";

Pattern pattern = Pattern.compile(regex);
Matcher matcher = pattern.matcher(email);
if (matcher.matches()) {
      System.out.println("Valid email!");
} else {
     System.out.println("Invalid email!");
   }
  }
 }
```

9. Explain how Java Regular Expression can be used for input validation. Write a regex patterns to validate an email address and describe how it work.

Java provides the Pattern and Matcher classes from java.util.regex packages to validate input using regular expression (regex).

Email Validation Example

```
^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-9]+\.[a-zA-Z]{2,}$
```

Explanation:

- ^ – start of String
- [a-zA-Z0-9._%+-]+ — One or more valid characters before e.
- @ – at symbol
- [a-zA-Z0-9.-]+ — Domain Name
- \. – Literal dot

Example: click event on dynamically added

```
<div id = "containen">
</div>
document.getIlementbyId('containen').addEventList-
ener('click', function(e){
if (e.target.tagName == 'BUTTON'){
alert('Button clicked: '+ e.target.textContent);
}
});
for (let i = 1; i<= 3; i++) {
let btn = document.createElement('button');
btn.textContent = 'Button' + i;
document.getElementById('containen').appendChil
(btn);
}
</script>
```

0 - A B → C → LIB
C D
3 -

```
document.getElementById("menu").addEventListener(
"click", function(e) {
if (e.target.tagName === "LI") {
alert("Clicked:" + e.target.innerText );
} });
```

8. What is event delegation in JavaScript and how does it optimize performance?

Event delegation in JavaScript is a techniqu where a single event listener is added to a parent element, instead of multiple listener on each child element.

It's useful because—

1) Better Performance
2) works with dynamic elements.
3) Cleaner Code.

7. How does the virtual DOM in React improve performance ?

Ans: React creates a virtual cop of DOM

On update, React:

1) Compares (diffs) old and new Virtual DOM.

2) Finds whats changed

3) Applies only the changes to real DOM

8. Event delegation in JavaScript

A technique where a single event listener is attached to a parent element to handles events from current and future child elements

```
<ul id ="menu">
<li> Home </li>
<li> About </li>
</ul>
```

```
public class CarParkingSystem {
public static void main(String [] arg) {
ParkingPool pool = new ParkingPool ();
new ParkingAgent (pool, 1). start ();
new ParkingAgent (pool, 2). start ();
new RegistrarParking ("ABC123", pool). start();
new RegistrarParking ("XYZ456", pool). start();
} }
```

Output:
```
Car ABC123 requested parking
Car XYZ456 requested parking.
Agent 1 parked car ABC123
Agent 2 parked car XYZ456.
```

6. Comparison between ~~DOM~~ DOM VS SAX.

Feature	DOM	SAX
memory	High (loads whole xml)	Low (reads line by line)
speed	Slower for big files	faster for large fil
Navigation	Easy (tree structure)	Hard
odification	Yes	NO
st for	Small xmL, editing	Large xmL

```
class RegisterParking extends Thread {
 private final String carNumber;
 private final parkingPool pool;
public RegisterParking(String carNumber, Parking
                                         Pool pool) {
 this.carNumber = carNumber;
 this.pool = pool; }
 public void run() {
    pool.addCar(carNumber); } }
class ParkingAgent extends Thread {
 private final ParkingPool pool;
 private final int agentId;
public ParkingAgent(ParkingPool pool, int agentId) {
  this.pool = pool;
  this.agentId = agentId; }
  public void run() {
   while (true) {
      String car = pool.getCar();
System.out.println("Agent " + agentId + " parked car
                              + car + ".");
try {Thread.sleep(500);} catch(Interrupted
                              Exception e) {}
   } }
 } }
```

Ans to the Ques No:5

Multithread based project

```
import java.util.*;
import java.util concurrent.*;
class ParkingPool {
    private final Queue <String> queue = new Linked
                                        List <>();
public synchronized void addCar(String car) {
queue.add(car);
System.out.println("Car "+ car+ " requested park-
ing. ");
notify(); }
public synchronized String getCar() {
while (queue.isEmpty()) {
try { wait();} catch (InterruptedException e)
{}
  return queue.poll();
}
}
```

Output: apple = 2
banana = 1
mango = 1

iii) Queue & Stack using PriorityQueue

```
import java.util.*;
public class PQStackQueue {
   static class Element {
   int val, order;
   Element (int v, int o) {val = v; order = o;}}
public static void main(String[]args) {
PriorityQueue<Element> stack = new PriorityQueue<>
((a,b) -> b.order - a.order);
PriorityQueue<Element> queue = new PriorityQueue<>
(Comparator.comparingInt (a -> a.order));
int order = 0;
  stack.add (new Element (10, order ++));
  stack.add (new Element (20, order ++));
  System.out. println("Stack pop: "+ stack. poll().val)
   queue.add (new Element (100,0));
   queue.add (new Element (200, 1));
System.out.println ("Queue poll : "+ queue.poll().val
   }}
```

4) i) find kth smallest element

```
import java.util.*;
public class KthSmallest {
public static void main(String [] args) {
List<Integer> list = Arrays.asList(8,2 , 5,1,9,4);
Collection.sort(list);
int k = 3;
System.out.println(k+ "rd smallest "+ list.get(k-1)
} }   Output: 4
```

ii) Word frequency using treemap

```
import java.util.*;
public class WordFreq {
  public static void main(String[] args) {
String text ="apple banana apple mango";
TreeMap<String, Integer> map = new TreeMap<>
for (String word : text.split(" "))
map.put(word, map.getOrDefault(word , 0)+
map.forEach((K ,V)-> System.out.print(K+" = "+V
}}
```

```
public class BankAccount {
  private String accountNumber;
  private double balance;
public void setAccountNumber(String AccountNumber){
if (accountNumber ==null }|| accountNumber.trim().isEmpty()
{
 throw new IllegalArgumentException("Can't be null");
} this.accountNumber = accountNumber; }
public void setInitialBalance(double balance){
if (balance <0){
    throw new IllegalArgumentException("Balance can't
                                          be negativ;
} this.balance = balance;
}
 public String getAccountNumber() {
 return accountNumber; }
public double getBalance(){
     return balance; }
public void deposit (double amount) {
     if (amount >0) this.balance += amount;
}
```

When to use interface —
i) You wants to define pure behaviour, not implementation.
ii) You want to use multiple inheritance of type.
iii) Classes are unrelated but share common capabilities.

3. How does encapsulation ensure data security and integrity? Show with an bank account class using private variables and validated methods such as setAccountNumber(String) etc.

Encapsulation is a key principle in object-oriented programming that hide internal data. It helps data security, data integrity, maintainability

2. Compare abstract classes and interfaces in terms of multiple inheritance.

Feature	Abstract class	Interface
multiple inheritance	not supported	fully supported
extends	clssA extends classB	ClassA implements x,y,z
Code reuse	Can have method bodies and member variable.	Java 8+ can have default and static methods.
State	Can have instance variables.	Only constants (public static final)
Constructor	Yes (can initalize field)	No constructor allowed.

When to use an abstract class:

i) You want to provide base functionality and shared states.
ii) You need constructor or non-static instance variables.
iii) You expected closely related classes with an "is-a" relationship

Since both classes in the same package the prot-
ected member message is accessable in child class

Protected in different classes.

```
package pack1;
public class Parent {
    protected String message = " Hello";
}
package pack2 ;
import pack1.parent ;
public class Child extends Parent {
public void showMessage () {
System.out.println("Access from child : "+message);
}
 public static void main(String[] args) {
 Child c = new Child ();
 c. showMessage (); }}
```

A sub class in a different package can acce
protected member of the parent class only throug
inheritance , not through the object of the
parent.

Imrose Aharaf
IT-22055

1. Demostrate how a child class can access a protected member of its parent class within the same package.

Accessing protected member in same package |
parent.java

```
package pack1;
public class Parent {
      protected string message = "Hello from p";
child.java
package pack1;
public class child ~~acce~~ extends Parent {
   public void showMessage () {
      System.out.println(" child accessed: "+message);
}
   public static void main (string [] args) {
        child c = new Child ();
        c.showMessage ();
   }
}
```