IMRAN HAMID
ID: IT-22010

Ans. to the qus. No-1

01. 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 different package?

Ans: In Java, a protected member of a class can be accessed by any subclass in the same package. When the child class is in the same package as it parent, it can directly access the protected members inherited from the parent class.

```
// parent.java
package mypackage;
public class parent {
    protected String message = "Hello from parent";
}
```

```
// child.java
package mypackage;
public class child extends Parent {
    public void showMessage() {
        System.out.println(message);
    }
    public static void main(String[] args) {
        child obj = new child();
        obj.showMessage();
    }
}
```

Here both Parent and child class belong to the same package named mypackage. The variable message in parent class is declared as protected. This allow the child class to access it direct. In child class, the method showMessage() prints the value of the message which it

inherits from the Parent. Since they are in the same package, the protected member is accessible to the child class.

If I use different package for child class:

```
// file : parent.java
// package : parentpackage
package Parentpackage;
public class panent{
  protected String msg = 'Hello';

// file: child.java
package childpackage;
import parentpackage.panent;
public class child extends panent{
    public void showmsg(){
      system.out.print(msg);
  }
```

```
public static void main (String[] args){
    Child obj = new Child();
    obj.showmsg();
    Parent p = new Parent();
    sysos(p.message() // this is not allowed
}
```

The child class is in a different package. It can access the message variable directly because it inherits it. But if you create a Parent class object (P) inside the child class and try to access msg using p.msg, it will cause a compile time error because protected members are not accessible through parent objects in different package. So in diff. packages, protected members can be accessed only by inheritance, not through parent class reference

Ans. to the ques. No. 2

02. Compare Abstract class and interfaces in terms of multiple inheritance:

Feature	Abstract class	Interface
i) Multiple inheritance support	Not supported	Supported
ii) Syntax for inheritance	class A extends B { }	class A implements B, C { }
iii) Use case	used when classes shared common code	used to define a common contract for unrelated classes
iv) Conflict Handling	No conflict	If multiple inheritance have same default methods, class must override it.

Use Abstract Class when:

i) You want to share common code among related classes.

ii) These classes have an "is-a" relationship.

iii) You want to define some default behaviour and leave the rest to be implemented by sub

Use Interface when:

i) You want to achieve multiple inheritance

ii) You want to define a contract that many unrelated class can implement.

iii) You don't need to store state.

Ans. to the ques. No. 8

Encapsulation is a core concept of OOP that ensures data security and integrity by restricting direct access to a class's internal variable. Instead of accessing fields directly, data is accessed and modified through public method (setters and getters) with validation logic to prevent invalid inputs.

By making variable private, they're hidden from outside classes, allowing the class to fully control how its data is handled and updated.

Example: Bank Account Class using Encapsulation

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

public void setAccountNumber (String accNum) {
    if (accNum != null && !accNum.trim().isEmpty()) {
        this.accountNumber = accNum;
    }
```

```
else {
    system.out.print("Invalid account");
  }
}

public void setInitialBalance (double amount) {
    if (amount >= 0) {
    this.balance = amount;
  }
  else {
    System.out.println("Initial balance can't be negative");
  }

public String getAccountNumber () {
    return accountNumber;
  }

public double getBalance () {
    return balance;
  }
}
```

Ans. to the ques. No- 4

(i) Find Kth smallest element in the arraylist.

Code:

```
import java.util.*;
public class KthSmallest {
    public static void main (String[] args) {
ArrayList <Integer> list = new ArrayList <> (Arrays.asList
                                    (8,3,5,1,9));
        int K = 3;
        Collections.sort(list);
        System.out.println (K + "th smallest: " + list.get(K-1));
    }
}
```

(ii) TreeMap to store word frequencies in Text.

Code:

```
import java.util.*;
public class WordFrequency {
    public static void main (String[] args) {
    String text = " apple banana apple orange apple ";
    String[] words = text.split(" ");
TreeMap <String, Integer> map = new TreeMap <> ();
```

```
for (String word : words) {
    map.put (word, map.getOrDefault (word, 0) + 1);
}
system.out.println (map);
}
}
```

(vi) Hashmap to store employee IDs and departments.

```
import java.util.*;
public class EmployeeMap {
    public static void main (String[] args) {
    HashMap <Integer, String> empDept = new HashMap<>();
    empDept.put (101, "HR");
    empDept.put (102, "IT");
    System.out.println (empDept);
    }
}
```

Ans. to the ques. No-5

Complete Code:

```
import java.util.concurrent.*;
class RegistarParking {
    private final String carNumber;
    public RegistarParking(String carNumber) {
        this.carNumber = carNumber;
    }
    public String getCarNumber() {
        return carNumber;
    }
}
class ParkingPool {
private final BlockingQueue<RegistarParking> queue = new LinkedBlocki
    public void addRequest(RegistarParking request) throws
    InterruptedException {
    queue.put(request);
    System.out.print("Car" + request.getCarNumber() +
                    " request parking.");
    }
```

```
public RegisterPanling getRequest() throws InterruptedException
    return queue.take();
  }
}

class panlingAgent extends Thread {
    private final PanlingPool pool;
    private final int agentId;
{
  public void run() {
   try {
     while (true) {
       RegisterPanling request = pool.getRequest();
       System.out.println("Agen" + agentId + request.getCanNumber());
       Thread.sleep(500);
     }
   } catch (InterruptedException e) {
       System.out.println("Agent" + agentId + "Stop");
     }
   }
}
```

```
public class MainClass {

public static void main (String[] a) throws

InterruptedException {

ParkingPool pool = new ParkingPool();

for (int i=1 ; i<=2 ; i++){

  new ParkingAgent (pool, i).start();

}

String[] cars = {"ABC123", "Bugati", "Toyota"};

  for (String car : cars) {

    new Thread (() -> {

      try {

        pool.addrequest (new RegisterParking (car));

    } catch (InterruptedException e) {

      e.printStackTrace();

    }

  }).start();

  Thread.sleep (500);

}

}
```

Ans. to the ques. No- 6

How java handles XML data using DOM and SAX parsers:

Ans:

DOM Parsers:

i) Reads the entire XML file and loads it into memory as a tree structure.

ii) allows random access to any part of the XML doc.

iii) Good for modifying or navigating the doc, multiple times.

SAX parsers:

i) Reads the XML file sequentially using events

ii) Doesnot store the whole document in memory

iii) You write handler methods to process data on the fly.

Scenario where SAX preferred than DOM:

Ans: Suppose you have a very large XML file (1GB) containing thousands of customers record, and you only wants to extracts names of customer from

Dhaka without loading the entire file. In this case SAX parser is preffered, as it uses less memory and is faster reading & filtering data.

Comparison:

Feature	DOM	SAX
Memory usage	High - entire XML is load into memory	low - reads one element at a time.
Processing speed	Slower for large files	Faster for large files.
Ease of use	easier to code and understand	requires more complex code.
Best for:	Small to medium XML files	Large XML files

Ans. to the ques. No-7

How Virtual DOM improve performance in React:

- In React when a compment changes:

i) React creates a new virtual DOM tree

ii) compare the new tree with previous virtual DOM tree using a diffing algorithm.

iii) It finds the minimal set of changes.

iv) Only these specific changes are applied to the real DOM.

This reduces the number of DOM manipulations, which improves speed and performance.

Diffing Algorithm:

Determines what has changed.

- It compares nodes by type and key.
- If the type is the same → it updates only the changed attributes.
- → If it is different → it replaces the node.

Comparison:

Features	Traditional DOM	Virtual DOM
i) DOM updates	Direct & Frequent	Batched & Optimized
ii) Performance	Slower with frequent UI updates	Faster due to minimal DOM changes
iii) Memory usage	Low	Slightly higher

Ans. to the ques. No. 8

Event Delegation: Event Delegation is a technique in JavaScript where you attach a single event listener to a parent element instead of multiple child elements. This listener uses event bubbling to catch events from child elements.

How it optimizes performance:

- Instead of adding many event listeners to individual elements, event delegation uses just one listener on the parent.

It works even if the child ~~mana~~ elements are dynamically added later, because the event bubbles up from the child to the parent.

This make application faster & more efficient.

Ans. to the ques. No. 9

Java Regular expression (regex) are used for input validation by defining a specific pattern that user input must follow. Java provides the pattern and Matcher classes from the java.util.regex packages to apply these patterns.

By compiling a regex using Pattern.compile() and matching it against input using Matcher.matches() Java can check if the input is valid. This helps ensure that only correctly formatted data is accepted before processing.

Regex pattern:

String emailRegex = "^[A-Za-z0-9+_.-]
+@[A-Za-z0-9.-]+$";

Code:

```
import java.util.regex.*;
public class EmailValidator {
    psvm () {
    String email = "example123@gmail.com";
    String emailRegex = "^[A-Za-z0-9+_.-]+@[A-Za-z0
                                          +$
        Pattern p = p.compile (emailRegex);
        Matcher m = p.m (email);
    if (m.matches()) {
        sys ("Valid");
    } else {
        sys ("Invalid");
    }
}
```

P.compile() creates a regex pattern

.m() compares the string with the pattern

matches() returns true if matches with regex

Ans. to the ques. No-10

Custom Annotations in Java are user defined annotations that allow developers to add metadata to classes, methods or fields. They do not change behaviour directly, but they can be read at runtime using reflection to influence the program's behaviour dynamically.

01. Define Custom Annotations :

```
import java.lang.annotation.*;
@Retention (RetentionPolicy.RUNTIME)
@Target (ElementType.METHOD)
public @interface RunMe {
    String value() default "Run this method";
}
```

02. Use the annotations :

```
public class MyClass {
    @RunMe ("say Hello")
    public void hello() {
        System.out.print ("Hello!");
    }
}
```

03. Process with reflection:

```
import java.lang.reflect.*;
public class Main {
    public static void main (String[] args) throws Exception {
        MyClass obj = new MyClass();
        for (Method m : MyClass.class.getDeclaredMethods()) {
            if (m.isAnnotationPresent(RunMe.class)) {
                RunMe ann = m.getAnnotation(RunMe.class);
                System.out.println("Annotation: " + ann.value());
                m.invoke(obj);
            }
        }
    }
}
```

Output: Annotation: Say Hello

Hello!

This shows how custom annotations can trigger behaviour at runtime using reflection.

Ans. to the ques. No- 11

The Singleton pattern ensures that only one instance of a class is created and provides a global access point to it.

It solves the problem of multiple objects accessing shared resources inconsistently.

It ensures a single instance by:

- Making the constructor private,
- Using a static method to create and return the same instance every time.

How thread safety can be achieved in Singleton implement?

Ans: In multithreaded environments, without precautions, multiple threads may create multiple instances at the same time.

To ensure thread safety, the following methods are used:

a) Synchronization Method:

- Makes the getInstance() synchronized, so only one thread can execute it at a time.
- Simple but can be slow due to locking every time.

(b) Double Checked Locking:

- Check if the instance is null before and after sync.
- Reduces performance cost by sync. only once when needed.

(c) Static Inner Class:

- Uses a nested static class to hold the instance.
- The class is only loaded when getInstance() is called, making it thread-safe, lazy loaded and efficient without synchronization.

All these techniques ensure that only one object is created when accessed by multiple threads.

Ans. to the ques. No. 12

JDBC helps java applications communicate with relational databases. It allows executing SQL commands and retrieving results. To perform select query, first the JDBC driver is loaded, a connection is made using DriverManager, then a statement or prepared statement is created. The query is executed and results are read using ResultSet. Error handling is done by using try-catch, and resources are closed in the finally block to avoid memory leaks.

Steps involved:

i) Load JDBC Driver
ii) Connect to DB
iii) Create Statement
iv) Execute Query
v) Use ResultSet to fetch Data
vi) Handles error with try-catch
vii) Close resources finally

Ans. to the ques. No-13

How do Servlets and JSPs work together in a web app following MVC architecture?

Ans: Servlets and JSPs work together to separate the applications logic, data and presentation.

The servlets acts as the controller, which handle

user request, processes input and decides which view to show. The JSP acts as the view, which display the output to the user. The Model is usually a Java class or bean that contain data or business logic.

Brief use case:

Consider a simple web app that displays a list of students, The Java Class student acts as the model which holds the students data like ID, Name. When a user request the student list, the servlet acts as the controller by handling the request, creating or retrieving a list of student objects and setting this list as an attribute in the req. object. The servelt forward the req. to JSP Page, which acts as the View. The JSP accesses the student list from the request iterates over it and display the students details in an HTML format. This way they work.

Ans. to the que. No-14

Life cycle of a servlet:

The life cycle of a servlet is managed by the servlet container (like Tomcat) and includes three main stages. creation, service and destruction. When a servlet is first requested, the container loads the servlet class and calls the init() method. This is followed by the service method() which handles client request. Finally when the servlet being removed from the memory, the container calls the destroy() method to clean up the resources.

Roles of Methods:

init(): called once when the servlet is first created. Used for initialization tasks like opening database conn. or reading configurations.

service(): called every time the servlet receives a request. It determines the HTTP method and forward the request to doGet() and doPost().

Destroy(): called once before the servlet is removed from the memory.

Concurrent Request and Thread safety issues:

By default, only one instance of a servlet is created, and it is shared by multiple threads. When many clients send req. at the same time, the servlet service() called by multiple threads concurrently. This can lead to thread safety issues if shared resources are modified without synchronization. To prevent problem developers must use synchronization, avoid mutable shared state or consider design patterns that isolate request data per thread.

Ans. to the ques. No-15

In a servlet only one instance is created by the servlet container and it handles multiple client requests using separate threads. If multiple threads access or modify shared resources, such as instance variables, data inconsistency or race condition may occur. This is because one thread might change a variable while another is reading or updating it, leading to unexpected results or corrupted data.

Example:

Suppose we have a servlet that uses an instance variable to count the number of visitors. If two users access the servlet at the same time, both threads might read the same value before it is update, resulting in the same visitor number being shown to both users.

To prevent such issues, we can synchronize the critical section where the shared variable is accessed. This ensures that only one thread at a time can execute the code inside the block which makes the update to thread safe.

Ans. to the ques. No-16

The MVC pattern separates an application into three components to organize code more efficiently. In a Java Web application, the Model represents the data and business logic, View the user interface and controller handles user input and application flow. This separation helps manage code better by assigning distinct responsibilities to each part.

For example, in a student reg. system, the model could be a Java Class storing student detail and performing validation. The Controller, implemented using a servlet, would handle HTTP requests, validate from data and interact with the Model. The View, built using JSP, would present the registration form and success message to the user.

Advantages:

The MVC pattern provide significant advantages. It improves maintainability because developers can modify the UI without changing the logic and it enchances scalability since different components can be updated or expanded ~~significan~~ independently. This makes the application cleaner, easier to test, and adaptable for future growth,

Ans. to the ques. No-17

In Java EE application following the MVC pattern, a servlet acts as the controller by managing the flow between the Model (Java classes) and the view (JSP). When a user sends a, the servlet processes the request, interacts with the model to perform business logic or fetch data, and then forwards the data to a JSP page for rendering the response. The servlet uses request.setAttribute() to pass data and then uses RequestDispatcher.forward() to transfer control to the JSP. This approach clearly separates business logic from presentation layer.

Brief Example:

```
// Servlet (Controller)
@WebServlet("/student")
public class StudentServlet extends HttpServlet {
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    // Model: create data
    Student student = new Student(1, "RMONO");
```

```
// pass data to the view
request.setAttribute("studentData", student);
// forward to JSP
request.getRequestDispatcher("student.jsp").forward(request, response);
}
}
```

JSP

```
<!-- JSP (view) - student.jsp -->
<%@ page import = "your package.Student" %>
<%
    Student s = (Student) request.getAttribute("studentData");
%>
<h2> Student Info : </h2>
<p> ID : <%= s.getId() %> </p>
<p> Name : <%= s.getName() %> </p>
```

This example shows the servlet controlling the application flow by fetching data from the model and forwarding it to the JSP for display. This structure ensures separation of concerns and better maintainability.

Ans. to the ques. No-18

Compare and constrast cookies, URLRewriting and HTTP session:

Feature	Cookies	URLRewriting	HTTP Session
i) Data Location	Stored on Client Browser	passed in the URL	Stored on server
ii) Visibility	visible to the user	Visible in browser address bar	Hidden from user
iii) Store Objects	No	No	Yes
iv) Security	Moderate	low	High
v) Performance	Lightweight	URL can become long	uses server memory.

Cookies:

Advantages: (i) simple to implement

(ii) Works accross multiple pages and servers if configured.

Limitations: (i) can be disabled by the user

(ii) Data is exposed in the browser.

Use case: remembering user preferences or tracking user sessions when cookies are enabled.

(ii) URL Rewriting:

Advantages: (i) Works even if cookies are disabled.

(ii) Easy to implement in simple applications.

Limitations:

(i) Expose data in the URL

(ii) Not ideal for large data

Use Case: cookies turned off and basic session track need.

(iii) Http Session:

Advantages: Easy to use with Java API

(ii) supports storing objects.

Limitations: (i) uses server memory

(ii) May not work properly if client disables cookies

Use case: E-commerce app, login systems etc.

Ans. to the ques. No-19

In a web application when a user logs in, the servlet creates an Http session object using request.getSession(). This session stores user-specific data like username on the server, and the client receives a session ID. This ID is sent with each request so the server can identify the user.

The session remains active until it times out automatically or is manually invalidated using session.invalidate() such as during logout. This prevents sensitive data from lingering.

To enhance security, session IDs should be protected with Https and secure cookies and not exposed in URLs. Timeout and invalidation help prevent unauthorized access on shared devices.

Ans. to the ques. No-20

In Spring MVC, when a browser sends an HTTP request the DispatcherServlet receives it and forwards it to the right component. The @Controller annotation marks a class as a controller to handle requests, and @RequestMapping maps methods to specific URLs. This tells spring method to call for a request.

The Model object passes data from the controller to the view, storing attributes like user info or msgs. The view uses this data to display output. This separation keeps business logic in the controller and presentation in the view, following MVC pattern.

Login Flow Example:

1. User submits the login form via /login
2. DispatcherServlet routes the request to a controller method like: @RequestMapping("/login");

3. The method checks user credentials and adds messages on user data to the model.

4. The method returns the logical view name, like "welcome".

5. The view resolver maps it to welcome.jsp, which displays the result.

Ans. to the ques. No-21

In Spring MVC, the DispatcherServlet acts as the front controller, handling all incoming HTTP request. It uses HandlerMapping to find the appropriate @Controller method based on the URL pattern.

After the controller processes the request, it returns a logical view name and model data. This DispatcherServlet then uses a ViewResolver to map the view name to an actual view file (like JSP or Thymeleaf)

finally the view is rendered and sent back to the user. This structure separates concerns and ensures smooth coordination between the controller, model, view in the Spring MVC framework.

Ans. to the ques. No-22

In JDBC, prepared statement improves both performance and security compared to statement.

Performance: PreparedStatement allows the SQL query to be precompiled by the database, so it can be executed multiple times with different parameters efficiently.

Security: It helps prevent SQL injection attacks by automatically escaping characters in input values.

Example: Inserting a Record using prepared statement.

```
import java.sql.*;

public class InsertExample {

    public static void main (String[] args) {

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

        String user = "root";

        String pass = "password";

    try ( Connection c = DriverManager.getConnection (url, user, pass)) {

        String sql = "insert into students (name, email) values (?,?)";

        ps.setString (1, "Imran");

        ps.setString (2, "imran@gmail.com");

        int rows = ps.executeUpdate();

        System.out.println (rows + " row(s) inserted.");

    } catch (SQLException e) {

        e.printStackTrace();

    }

}

}
```

Ans. to the ques. No-23

In JDBC a resultset is an object that stores the result of SQL select Query executed using a statement or PreparedStatement. It acts like a table in memory, where each row can be accessed one by one.

The next() moves the cursor to the next row in the result set and returns true if another row exists.

The getString("ColumnName") method retrieves a column's value as a string.

The getInt("Column Name"), method retrieves an integer value from the specified column

Example: Retrieving Data From MySQL

```
import java.sql.*;
public class ResultSetExample {
    psvm (String[] args) {
    String url = "jdbc:mysql://localhost:3306/mydb";
    String user = "root",
    String pass = "demon");
try (Connection c = DriverManger.getConnection (url, user, pass));
    String sql = "Select id, name from students";
    Statement s = c.createStatement();
    ResultSet rs = s.executeQuery (sql);
while (rs.next()) {
    int id = rs.getInt ("id");
    String name = rs.getString ("name");
    Syos ("ID: " + id + ", "name: " + name);
}
} catch (SQLException e) {
    e.printStackTrace();
}
}
```

Ans. to the ques. No. 24

How JPA manages mapping between Java Objects and Relational Table:

⇒ JPA is a specification that allows you to map Java classes to relational database tables, handling all CRUD operations automatically.

@Entity → Declares that the class is a table in the Database

@ID → Marks the primary key field

@GeneratedValue → Auto generate primary key values

Feature	JPA	JDBC
Abstraction	High Level, Object Oriented	Low Level, Manual SQL needed
Caching	Supported	Not built in
Query support	JPQL	Raw SQL only
Code size	Less boilerplate	Require verbose code.

Ans. to the ques. No - 25

Action	persist()	merge()	remove()
(i) Action	Insert	Update/ attach	Delete
(ii) Entity must be	New	Detached	Managed
(iii) Use case	Save a new object	update or attach a detached object	Delete an existing object

Ans. to the que. No. 21

01. Entity class : student.java

```
@Entity
public class Student{

    @Id
    @GeneratedValue (strategy = GenerationType.IDENTITY)

    private Long id;
    private String name;
    private int age;

    // setters & getters

}
```

02. Repository interface

```
public interface StudentRepository extends
        JpaRepository <Student, Long> { }
```

03. Controller : StudentController.java

```
@RestController
@RequestMapping ("/students")
public class StudentController {
```

```
@Autowired
private StudentRepository repo;

@PostMapping
public Student create(@RequestBody Student s){

    return repo.save(s);

}

@ReadAll
@GetMapping
public List<Student> readAll(){

    return repo.findAll();

}

@GetMapping("/{id}")
public Student read(@PathVariable Long id){

  return repo.findById(id).orElse(null);

}

@PutMapping("/{id}")
public Student update(@PathVariable Long id, @RequestBody
                                                Student s){

    s.setId(id);
    return repo.save(s);
}
```

```
@DeleteMapping("/{id}")
public void delete(@PathVariable Long id){
    repo.deleteById(id);
  }
}
```

(4) Application.properties :

```
spring.datasource.url = jdbc:mysql://localhost:3306/project
spring.datasource.username = root
spring.datasource.password = demon
spring.jpa.hibernate.ddl-auto = update
```

That's it - fun CRUD using Spring Data JPA.

Ans. to the ques. No. 27

How spring boot simplies Restful Services:

- Auto configure everything (no boilerplate XML)
- Embedded server (Tomcat)
- Easy Annotations for REST APIs
- Built in JSON handling via Jackson

Example: REST Controller

```
@ RestController
@ RequestMapping ("/api/users")
public class UserControllers {
    @ GetMapping
    public List <usr> getUsers () {
    return List.of (new User (1, "AKil"), new (2, "Sama"));
    }
    @ post Mapping
    public User createUser (@ RequestBody User user) {
        return user;
    }
}
```

JSON example :

Request (Post) :

```
{
"id": 1,
"name": "Akil"
}
```

Response (GET / POST) :

```
[
{"id": 1, "name": "Akil"}
]
```

Ans. to the ques. No-28

Aspect	@RestController	@Controller
i) Use Case	Rest API's	Web pages
ii) Returns	Direct JSON	View pages (HTML)
iii) Example return	return List<Book> → JSON	return "home" → loads home.html

02. RESTful Endpoints for Books in a Library System

Operation	HTTP Method	Endpoint	Description
Create Book	Post	/books	Add a new book
Read all books	GET	/books	Get list of books
Read one book	GET	/books/{id}	Get book by ID
Update book	PUT	/books/{id}	Update book info by ID
Delete book	DELETE	/books/{id}	Delete book by id

Ans. to the ques. No – 29

How MAVEN manages Dependencies and build Lifecycle:

Dependencies:

- Maven uses the pom.xml file to declare libraries.
- It auto downloads and manages versions from Maven Central
- Ensures all libraries are compitalle and avoid "JAR hell".

Build Lifecycle:

Standard phase like:

compile → compiles Java code

test → Runs tests

package → Builds .jar or .war

install → installs to local rep.

deploy → Deploys to remot repo.

How spring boot Stanter Dependencies help:

- spring-boot-stanter-web → brings Tomcat + JSON + REST + MVC
- spring-boot-starter-data-jpa → brings Hibernate + JPA
- spring-boot-stanter-test → brings JUnit + Mockito

These are no need to add manually.

Ans. to the que. No. 32

Project Name: Book_Recommended_system

Tech Stack: Spring Boot, Spring web, Spring Data JPA, Thymeleaf, MySQL

Main Features:

- Role based access : Admin and User
- Admin can : Add, View, Delete books
- User can : View and search books
- Book search by keyword or genre.

Key Code Snippets:

1. Admin login Logic (AdminLoginController.java)

```
@PostMapping("/login")
public String processLogin(@RequestParam String password,
                           HttpSession session) {

    Admin admin = adminRepository.findFirstByOrderByIdAsc();

    if (admin.getPassword().equals(password)) {

        session.attribute("role", "admin");
```

return "redirect:/books";

}

else {

return "adminlogin";

}

}

02. AddBook (BookController.java)

@PostMapping ("/books/add")

public String addBook (@ModelAttribute Book book, HTTPSession ses

{

if (!"admin".equals (session.getAttribute ("role")))

return "redirect:/books";

bookService.addBook (book);

return "redirect:/books";

}

03. Book Search (BookController.java)

```
@GetMapping("/search")
public String searchBooks(@RequestParam String keyword,
                          Model model) {
    List<Book> books = bookService.searchBooks(keyword);
    model.addAttribute("books", books);
    return "search";
}
```

GUI (Thymeleaf HTML)

01. entry.html : Role section

```
<a href="/admin/login"> Login as Admin </a>
<a href="/books"> Enter as User </a>
```

02. adminLogin.html : Password form

```
<form th:action="@{/admin/login}" method="post">
    <input type="password" name="password" />
    <button type="submit"> Login </button>
</form>
```

03. index.html : Book list with Add / Delete

```
<li th:each="book: ${books}">
  <strong th:text="${book.title}">Title</strong>
  <a th:href="@{'/books/delete/' + ${book.id}}"
     th:if="${role == 'admin'}">Delete</a>
</li>
```

04. addbook.html : Add form

```
<form th:action="@{/books/add}" th:object="${book}"
      method="post">
  <input th:field="*{title}" />
  <button type="submit">Add Book</button>
</form>
```

Also Database Tables:

(i) Admin (id, password)

(ii) Book (id, title, author, genre, imgurl)