Answer to the question no: 1

In Java protected members are accissible:

1. ~~Dir~~ Directly from parent class, while in same package
2. By importing the parent class from another package, while in different package.

1. Within same package:

```
package package_1;
public class Parent {
    protected string Message = "Hello child.";
}
```

```
package package_1;
public class child extends Parent{
    public void show(){
        System.out.println(Message);
    }
    public static void main(String args[]){
        Child c = new Child();
        c.show();
    }
}
```

2. Within different package :

```
package_1;
public class parent{
    protected string message = "Hello child.";
}
```

=

```
package_2;
import package_1.parent; //for diff class
public class child extends parent{
    public void show(){
        System.out.println(message);
    }
    public static void main(String args[]){
        child c= new child();
        c.show();
    }
}
```

Answer to the question no: 2

In Java abstract class and interface both used to achieve abstraction, but they behave differently when it comes to multiple inheritence:

(i) Abstract class does not support multiple inheritence, but interface supports that.

(ii) Its methods can be extended in a class, on the other hand interface are needed to be implemented.

(iii) Abstract class can have both abstract and concrete method, but interface methods are abstract by default.

When we use abstract class:

Abstract class are used to provide a common base for related classes and share some default methods.

When we use interface:

It is used in code when we need multiple inheritence means a class wanna extend multiple class properties in it.

```
interface A{
    void methodA();
}
interface B{
    void methodB();
}
public class Myclass implements A,B{
    public void methodA(){
        System.out.println("Method A");
    public void methodB(){
        System.out.println("MethodB");
    }
    public static void main(String args[]){
        Myclass c = new Myclass();
        c.methodA();
    }
}
```

Answer to the question no: 3

Encapsulation is the key concept of OOP that:

(i) Hides the ~~data~~ details of a class using private variables.

(ii) Allows control access to data through public methods.

The encapsulation ensure data security and integrity:

(i) Data security: variables can't be directly accessed

(ii) Data integrity: Only accepts valid data.

Here is a demo code to explain:

```
public class BankAccount{
    private String accountNumber;
    private double balance;
    //setter methods
    public void setAccountNumber(String accountNumber){
        if accountNumber==null||accountNumber.trim().isEmpty()){
            System.out.println("Invalid Account no.");}
        else{
            this.accountNumber=accountNumber;
        }
    }
```

```
public void setBalance(double balance){
    if (balance<0){
      System.out.println("Invalid balance");
    }
    else{
       this.balance = balance;
    }
}

public String getaccount(){
       return accountNumber; }

public double getBalance(){
       return balance; }

public static void main(String args[]){
    BankAccount a = new BankAccount();

    a.setAccountNumber("");  // invalid
    a.setInitialBalance(-500); // invalid
    a.setAccountNumber("ACC1234");
    a.setInitialBalance(10000);
System.out.println("Account : " + ~~account~~ a.getAccount());
System.out.println("Balance : " + a.getBalance());
}
}
```

Answers to the question no: 4

①

Kth smallest in arraylist:

```
import java.util.*;
public class KthSmallest{
    public static void main(String args[]){
        ArrayList<Integer> list = new ArrayList<>();
        list.add(5);
        list.add(2);
        list.add(9);
        list.add(1);

        int k = 3;

        Collections.sort(list);
        System.out.println((k+"th smallest element: "+list.get(k-1));

    }

}
```

(1)

gs two linked Lists are equal:

```
import java.util.LinkedList;
public class Linked_List{
    public static void main(String [] args){
    LinkedList<String> list1 = new LinkedList<>();
    LinkedList<String> list2 = new LinkedList<>();

     list1.add ("apple");
     list1. add ("banana");
     list1. add ("cherry");

     list2.add ("apple");
     list2.add ("banana");
     list2. add ("cherry");

     if (list1. equals(list2)) {
        System.out.println(" They are equal."); }
     else{
        System.out.println(" Not equal.");
        }
    }
}
```

(VI)

HashMap employee id dept:

```
import java.util.HashMap;
import java.util.Scanner;

public class Employee{
    publis static void main(String [] args){
    HashMap<Integer, String> employeeMap = new HashMap<>();
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the numbers of employees: ");
    int n = sc.nextInt();

    for(int i = 0; i<n; i++){
        System.out.println(Enter Employee ID: ");
        int empID = sc.nextInt();
        sc.nextLine();

        System.out.println("Enter Department: ");
        String dept = sc.nextLine();

        employeeMap.put(empID, dept);

    }
```

~~System.out~~

```
sc.nextLine();
for(Integer id : employeeMap.keyset()){
    System.out.println("ID: "+id+" → Dept: "+employeeMap.
                                                get(id));
}
}
}
```

Answer to the question no: 6

Java handles XML data using two main types of parsers:

① How DOM parser handles:

- Loads the entire XML document into memory as tree.
- Allows random access to any part of the document.
- Used via classes like DocumentBuilder, Node, Element etc.

2. How SAX parsen handles:

- Parses the XML file sequentially.
- Triggens callbacks (like: startElement(), characters())
- Does not load the entire XML into Memory.

Comparison with nespect to some propenties:

Feature	DOM	SAX
Memory usage	High (entire file in memory)	Low (Process Line by Line)
Process speed	Slowen for large file.	Fasten for large file.
User case	Easien for complex document	Handles eenents manually.

- When processing a large log file stored as XML and user want to extract specific events, we choose SAX over DOM.

(i) No need to load entire file.

(ii) Only intenested in specific elements.

(iii) Fasten and nesounce efficient.

Answer to the question no: 7

React uses Virtual DOM to optimize and speed up UI rendering by minimizing direct manipulations of the real DOM, reducing expenses and increasing speed.

Traditional DOM vs ~~Rea~~ Virtual DOM:

Feature	Traditional DOM	VDOM
Update method	Direct manipulation	virtual representation
Performance	slower for large update	Faster for ~~minial update~~ minimal update
Re-rendering	Updates the whole VI	Update only changed parts.
Ease of use	Manual update logic required	Automatic.

Working of Diffing Algorithm:

1. React renders a component and builds a new virtual DOM tree.
2. Compares new VDOM to previous one.
3. Calculates minimal changes.
4. Applies only those changes to real DOM.

Example:

React before:

```
function Greeting(){
    return <h1>Hello, Alice!</h1>;
}
```

After change:

```
function Greeting(){
    return <h1>Hello, Bob!</h1>;
}
```

Answer to the question no: 8

Event delegation is a technique where instead of adding listeners to multiple child elements, a single listener added to parent element attached and catched from children.

How it optimize performance:

(i) There are fewer event listener
(ii) No need to re-build listeners
(iii) centralized logics are easier to manage.

Example:

HTML:

```
<ul id = "itemList" >
  <li> Item 1 </li>
  <li> Item 2 <li>
<ul>

<button id = "addItem" > Add Item </button>
```

JavaScript with delegation:

```
const itemList = document.getElementById("itemList");
const addItem = document.getElementById("addItem");
let count = 3;

itemList.addEventListener("click", function(event){
    if (event.target.tagName === "LI"){
        alert("You clicked: " + event.target.textContent);
    }
});

addItem.addEventListener("click", function(){
    const newItem = document.createElement("li");
    newItem.textContent = "Item" + count++;
    itemList.appendChild(newItem);
});
```

Answer to the question no: 9

Java provides regex support through java.util.regex package :

(i) Form input validation (email, phone etc)
(ii) Pattern matching
(iii) Data extraction

There are 'pattern' and 'Matcher' key function to compile and check the input.

Validation of Email using Regex:

```
import java.util.regex.*;
public class EmailValidaton{
    public static void main(String [] args){
        String email = "it22011@mbstu.ac.bd";
        string emailRegex = "^[A-Za-z0-9+_.-]+@[A-Za-z0-9.-]+$";
        Pattern p = Pattern.compile(emailRegex);
        Matcher m = p.matcher(email);
```

```
if (matcher.matches()) {
    System.out.println("Valid.");
  }
  else {
    System.out.println("Invalid.");
  }
}
}
```

Answer to the question no: 10

Custom annotations in java allow developers to define metadata that can be attached to classes, method etc which can be read or processed at runtime.

They can be used to:

(i) Add metadata for configuration, validation etc.

(ii) Improve code readability.

(iii) Enable frameworks.

Designing a custom annotation @RunImmediately:

Defining:

```
import java.lang.annotation.*;
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface RunImmediately{
    int times() default 1;
}
```

Annotation methods:

```
public class MyService{
@RunImmediately(times = 3)
    public void start(){
        System.out.println("starting..");
    }
    @RunImmediately
    public void init(){
        System.out.println("Initializing..");}
    public void stop(){
        System.out.println("stopping..");
    }
}
```

Using Reflection:

```
import java.lang.reflect.Method;

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

Answer to the question no: 12

JDBC is an API that allows java applications to communicate with relational database using SQL.

JDBC steps for SELECT query:

1. Load JDBC driver
2. Establish connection
3. Create statement
4. Execute query
5. Process result (ResultSet rs)
6. Close resources.

Error handling code:

```
import java.sql.*;
public class JdbcSelectExample{
    public static void main(String [] args){
        Connection conn = null;
        Statement  stmt = null;
        ResultSet  rs   = null;
```

```
try {
    class.forName("com.mysql.cj.d jdbc.Driver");
    conn = DriverManager.getConnection
            ("jdbc:mysql://localhost:3306/dbname","user","pass");
    stmt = conn.createStatement();
    rs = stmt.executeQuery("select * from users");

    while(rs.next()){
        system.out.println("User: "+rs.getString("username"));
    }

}
catch(Exception e) {
    System.out.println("DB error: "+e.getMessage());

}
finally {
    try{ if(rs!=null)rs.close();} catch(Exception ignored){ }
    try{ if(stmt!=null)stmt.close();} catch(Exception ignored){ }
    try{ if(conn!=null)conn.close();} catch(Exception ignored){ }

    }

}
```

Answer to the question no: 13

Servlet and Jsp working together as MVC:

(I) Model : holding busing on data logics.

(II) Controller: servlet handles request and forwards to view.

(III) View : Jsp display data.

Use case as MVC: user (profile)

1. Model (user.java)

```
public class user {
    private String name;
    public user(String name) {this.name = name;}
    public String getName() {return name;}
}
```

2. Controller (UserServlet.java)

```
@WebServlet("/profile")
public class UserServlet extends HttpServlet {
    protected void doGet(HttpServletRequest req, ~~HttpServletR~~
                         HttpServletResponse res) throws
                         IOException, ServletException{
```

```
User user = new User("Alice");
req.setAttribute("user", user);
~~req.getRespon~~
req.getRequestDispatcher("profile.jsp").forward(req, res);

    }
}
```

3. View (profile.jsp)

```
<%@ page import = "yourpackage.user" %>
<jsp:useBean id = "user" type = "yourpackage.user" scope="request" />
<h1> Welcome, ${user.name}! </h1>
```

Answer to the question no: 14

A servlet's life cycle has three main phases controlled by init(), service() and destroy() methods.

① init():

① called once when the servlet is first loaded.

② Used for initialization

2. service():

① Called for every client request (Get, POST)

② Dispatches request to methods doGet() or doPost()

③ Handles core req and res.

3. destroy():

① Called once when server shuts down.

② Used for cleanup.

☐ How servlets handle concurrent req:

① A servlet instance handles multiple requests simultaneously by swapping threads calling service()

② Only one servlet instance exists by default.

☐ Thread safety issue:

① Instance variables can be accessed by multiple threads at the same time, leading data inconsistency.

② To avoid this:
- Use local variables
- Synchronize critical sections
- Use thread-safe objects.

Answer to the question no: 15

If multiple threads access shared variables, data can become inconsistent.
This is called Race condition Problem.

Example:

```
public class CountenServlet extends HttpServlet{
    private int count = 0;
    protected void doGet(HttpServletRequest req,
            HttpServletResponse res) throws IOException{
        count++;
        ~~req~~res.getWriter().println("Visitor count: "
                + count );
    }
}
```

The issus is: Two threads may read/update 'count' at same time.

solution using Synchronization:

```
Synchronized (this){
    count++;
}
```

This synchronizes this and fix the issue

Answer to the question no: 16

MVC in Java web application patterns separate concerns:

(I) Model: Handles data and logic

(II) View: JSP shows UI.

(III) Controller: Servlet manages flow.

From 'question 13' answer there is a student submit form shown, with concerns:

1. User submit form →
2. Servlet (controller) handles req →
3. calls Model to save student →
4. Forwards to JSP (view) for confirmation.

Advantages:

(I) Maintainability: Logic/UI separates, easier to update.

(II) Scalability: Add new features without touching all layers.

Answer to the question no: 17

A servlet acts as the controller, handling requests, invoking the model and forwarding data to the JSP (view) to render the response.

Example:

```
@WebServlet("/show")
public class StudentServlet extends HttpServlet {
    protected void doGet(HttpServletRequest req,
        HttpServletResponse res) throws ServletException,
                IOException {
        String name = "shakib";
        req.setAttribute("studentName", name);
        req.getRequestDispatcher("student.jsp").forward(req,res);
    }
}
```

JSP (view – student.jsp):

```
<p> student Name : ${studentName} </p>
```

Servlet handle logic, set data → forward to JSP → JSP display it.

Answer to the question no: 18

Here is a short comparison of session tracking methods:

Method	Advantages	Limitations	Ideal use case
cookies	-Simple to use -presistance storage	-user may disable cookies -size limit	Tracking returning users.
URL Rewriting	-works without cookies	-Manual rewriting expose data in URL	When cookies are disabled
Httpsession	-Automatic -stores object securely	-Needs url to track session id	Most web app needing user session state

Summary:

- cookies: Data stored in browser.
- URL Rewriting: session id append in URL.
- Httpsession: server stores session data.

Answer to the question no: 19

How HttpSession works accross requests:

(i) When a user logs in a session is created:

```
HttpSession session = request.getSession();
session.setAttribute("username", "shakib");
```

(ii) The session id is sent to the browser.

(iii) On the next request the browser sends back the session ID.

session timeout:

```
<session-config>
    <session-timeout>15</session-timeout>
</session-config>
```

or,

```
session.setMaxInactiveInterval(900); //15 mins
```

Invalidate manually:

```
session.invalidate(); // when log out...
```

Answer to the question no: 20

How Spring MVC handles Http requests:

spring MVC follows DispatcherServlet → controller → Model → View.

① @Controller : Marks a class as a controller to handle web req.

② @RequestMapping : Maps URLs to specific controller methods.

③ Model : Used to pass data to the view.

☐ Login flow Example:

① Browser's submits form to '/login':

```
<form action = "/login" method = "post" />
<input name = "username" />
<input name = "password" />
</form>
```

2. Controller handles the req:

```
@Controller
public class LoginController{
@PostMapping ("/login")
public string login(@RequestParam String username,
    @RequestParam string password, Model model){
    if ("admin".equals(username) && "1234".equals(Password)
        {
        model.addAttribute("msg","Login success");
        return "welcome";
        }
    else{
        medel.addAttribute("msg","Login failed");
        return "login";
    }
}
}
```

3. View (welcom.jsp or login.jsp) displays the

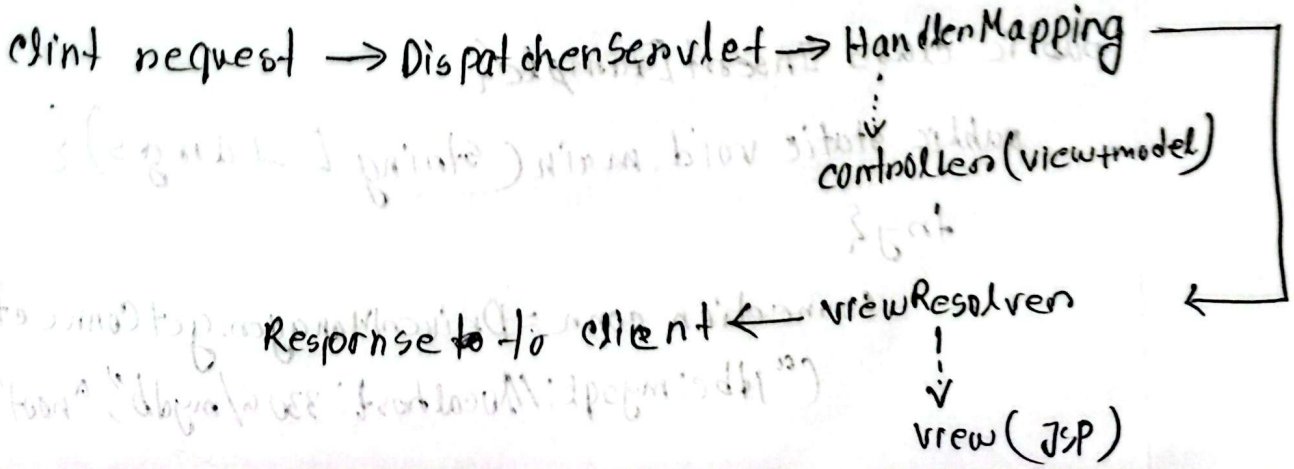
Answer to the question no: 21

Dispatcherservlet acts as the front controller, handling all http requests and coordinating the entire spring MVC workflow.

Request processing workflow:

1. Clint sends request
2. Handler Mapping
3. Controller executes
4. View Resolver
5. Dispatcherservlet renders view with the data and ~~ses~~ reponse to clint.

Interaction diagram:



Answer to the question no: 22

Prepared Statement improve performance and security over statement in JDBC. We can assume that by their comparison:

Features	Statement	Prepared Statement
Security	Prone to SQL injection x	Prevents sql injections ✓
Performance	Query compiled even time. x	Precompiled once ✓
Readability	Concatenation required	cleaner with ? placeholders.

Inserting record using Prepared Statement:

```
import java.sql.*;
public class InsertExample{
    public static void main(String[] args){
        try{
            Connection conn = DriverManager.getConnection
                ("jdbc:mysql://localhost:3306/mydb", "root", "password");
```

```
String sql = "Insert into employees (name, depantment)
                values(?,?)";
PreparedStatement ps = conn.prepare statement(sql);
ps.setString(1,"shakib");
ps.set string(2,"ID");
int rows = ps.executeUpdate();
System.out.println(rows+" row inserted.");
conn.close();
}catch(Exception e){
    e.printstackTrace();
}
}
}
```

Answer to the question no: 23

ResultSet is a Java object that holds data retrieved from a database.

It retrieves data by executing SELECT query using JDBC.

Common methods:

(i) next(): Moves the cursor to the next row.

(ii) getString("column"): Retrieves String data from a column.

(iii) getInt("column"): Retrieves int data from a column.

Example:

```
Connection con = DriverManager.getConnection
        ("jdbc:mysql://localhost:3306/mydb", "root", "password");
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("select id, name from
```

```
while(rs.next()){
    int id = rs.getInt("id");
    String name = rs.getString("name");
    System.out.println(id + " - " + name);
}
con.close();
```

Answer to the question no: 24

JPA uses annotations to map Java classes to databases table. It manages persistence without writing raw SQL.

Key annotations:

① @Entity - Marks the class as JPA entity (table).

② @Id - Specifies the primary key.

③ @GeneratedValue - Auto-generates primary key values.

Example:

```
Import jakarta.persistence.*;
@Entity
public class Employee {
    @ID
    @GeneratedValue(Strategy=GenerationType.IDENTITY)
    private int id;
    private string name;
    private string dept;
    //Getters and setters
}
```

It creates a table like:

```
create table Employee (
    id int auto_increment primary key,
    name varchar,
    dept varchar);
```

We choose JPA over JDBC because:

① JPA has minimal boilerplate and JDBC verbose with SQL.

② JPA has automatic mapping where JDBC has manual.

③ In JPA transition is build-in and JDBC has manual handling.

Answer to the question no: 25.

The EntityManager in JPA is used to manage the lifecycle of entities.

The key methods differ as:

① Persist (entity):

- Purpose: insert new entity in database.
- When to use: when saving a new record.
- Effects: Makes the object managed and scheduled for insertion at commit.

② menge(entity):

- Purpose: Updates an existing entity
- When to use: When updating data.
- Effect: Return managed copy of entity.

③ Remove(entity):

- Purpose: Deletes an entity from the database.
- When to use: To delete a record.
- Effect: Marks the entity for removal at commit time.

Answer to the question no: 27

SpringBoot simplifies Restful service by:

(i) Auto-configuring components

(ii) Reducing boilerplate via annotations.

(iii) Embedding JSON support

(iv) Providing easy-to-use annotations.

Simple REST controller:

1. Student Model:

```
public class Student{
    private Long id;
    private String name;

    // Getter and setter

}
```

2. REST controller:

```
@RestController
@RequestMapping("/student")
public class studentController {
    List<Student> list = new ArrayList<>();

@GetMapping
public List<Student> getAllStudents() {
    return list;
}

@PostMapping
public String addStudent(@RequestBody Student s){
    list.add(s);
    return "Student added.";
}
}
```

3. JSON input:

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

Answer to the question no: 28

@RestController vs @Controller in SpringBoot:

@Controller – Used for MVC web apps.

@RestController – Used for REST APIs.

@RestController = @Controller + @ResponseBody.

☐ ~~Rest~~ RESTful API structure for Library System:

HTTP Method	endpoint	Action
~~GET~~ GET	/books	Read all
GET	/books/{id}	Read one
POST	/books	create
PUT	/books/{id}	Update
DELETE	/books/{id}	delete

Answer to the question no: 29

Maven is a build automation and dependency management tool. It simplifies a spring Boot project lifecycle:

(i) Validate – checks the structure

(ii) Compile – compiles code

(iii) test – runs unit test

(iv) package – creates .jar or .war

(v) Install – adds to local Maven repo

(vi) deploy – uploads to remote repo

pom.xml ~~str~~ dependency structure:

```
<dependencies>
  <dependency>
    <groupID>org.springframework.boot</groupID>
    <antifactID> spring-boot-starter-web </antifactID>
  </dependency>
    :
</dependency>
```

starters:

spring-boot-starter-web → RESTAPI, tomcat

Spring-boot-starter-data-jpa → auto query

spring-boot-starter-security → Authentication

spring-boot-starter-test → JUnit, Mockito etc.

Answer to the question no: 30

Concise comparison of Maven and Gradle in context of Spring Boot projects:

Feature	Maven	Gradle
Language	XML	Groovy / Kotlin
Readability	Verbose	Concise
Performance	Slower	Faster
Dependency	Centralized via pom.xml	Flexible and customizable
Build customization	Less dynamic	More powerful scripting
Community support	Older	modern

Answer to the question no: 37

We structure a multi-module spring Boot application using Maven, we separate the modules as:

structure: