IT23024

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

Answer: In Java a protected member is variable, method or constructor. And these member can be accessed by-

i) Within same package any class can access protected member

ii) In different package only subclass can access protected members, and only through inheritance.

Let's look at both the scenarios →

Case1: Child class in same package

File: Parent.java

```
package mypackage;
public class format {
        protected string message = "Hellow from parent";
```

File: child.java

```
package mypackage;
  public void showMessage(){
      system.out.printIn(message);
  }
public static void main(string[] args){
      child c = new child();
      e.show message();
  }
}
```

Output:

Hello from parent

case2: child class in a diffenent package

File: parent.java

```
package mypackage;
public class parent{
  protected string msg = "Hellow I am Mimi
}
```

File: Child.java [In another package]

```
package child paekage;
import mypackage.parent
publie class child extends Parent{
    public void message(){
        System.out.println(msg);
    }
public static void main (string[] args){
    child c = new child();
    c.message();
}
}
```

Output:

Hellow I am mini.

So, from the example it is seen that protected member can be accessed both from the same package subclasses and from different package subclasses.

I123024

2. Compare abstract classes and interfaces in terms of multiple inheritance. When would you prefer to use astract class and when an interface?

Ans: Comparison Between Abstract Classes and Interfaces with respect to Multple inheritance is given below:

Object-oriented programming provides abstract classes and interfaces to archive abstraction. Although both are used to defined abstract behavion, they differ significantly in terms of multiple inheritance and usage.

Abstract Class

A class can inherit only one abstract class. Therefore abstract classes do not support multiple inheritance. This restriction exist to avoid ambiguity such as the diamond problem, where multiple paren classes contain the same method implementation.

Example:

```
class B extends A {}
class C extends A,D {}
```

IT23024

Interface

A class can implement multiple interfaces, which means interfaces support multiple inheritance. since interfaces mainly define method declarations, conflicts an avoided.

Example:

```
class A implements X, Y, Z { }
```

Key difference between Abstract class and Interface is given below:

Feature	Abstract class	Interface
Multiple inheritance	Not supported	Supported
Method implementation	Allowed	Allowed using default method
Instance variable	Allowed	Not allowed
Constructors	Allowed	Not allowed
Access modifier	Any	methods are default
Keyword used	extends	implements

IT23024

When to use an Abstract class

An abstract class should be used when:

1. There is a clean IS-A relationship between classes
2. Common fields and method implementations need to be shared
3. classes are closely related
4. Future modifications to base class behaviour are expected.

Example:

```
abstract class Vehicle{
    int speed;
   void start(){
    System.out.println("Vehicle started");
   }
  abstract void move();
}
```

When to use an Interface

An interface should be used when:

1. multiple inheritance is required
2. Only behavior needs to be defined
3. Classes are unrelated but share common functionality.
4. A loosely coupled design is preferred

Example:

```
interface Flyable{
    void fly();
}
class Bird implements Flyable{}
class Airplane implements Flyable{}
```

I123024

3. How does encapsulation ensure data security and integrity? Show with a BankAccount class using private variables and validated methods such as setAccountNumber(string), setIntialBalance(double) that rejects null, negative or empty values

Answer:

Encapsulation is a fundamental concept of object-oriented programming that binds data (variables) and methods together into a single unit and restricts direct access to the data. This is achieved by declaring variables as private and accessing them only through public, validated method.

Encapsulation ensures data security and data integrity in the following way:

1. Data Hiding (security):
   i) private variables cannot be accessed or modified directly from outside the class.
   ii) This prevents unauthorized or accidental change to sensitive data

2. Controlled Access

i) public setter method validate input before assigning values.

ii) Invalid data such as null, empty strings, or negative values are rejected.

iii) This ensure the object always remains in a valid and consistent state.

3. Prevention of Invalid states

i) Encapsulation enforces business rules.

ii) Any attempt to break these rules is blocked by validation logic

Example: BankAccount class Demonstrating Encapsulation

```
public class BankAccount {

    private string accountNumber;
    pnivate double balance;

    public void setAccountNumber (string accountNumber){
    if( accountNumber == null || accountNumber.trim().is
    Empty ()) {
    Throw new IllogalArgmentException ("Account number
    cannot be null or empty.");
    }
    this.AccountNumber = accountNumber;
}
```

```
public void setInitialBalance(double balance){
  if(balance <0){
throw new IllegalArgumentException("Initial balance
cannot be negative");
   }
this.balance = balance;
}
public string getAccountNumber(){
    return accountNumber;
  }
public double getBalance(){
      return balance;
    }
}
```

How this example Ensure Security and Integrity

1. The variables accountNumber and balance are private, so they cannot be accessed directly.
2. The method setAccountNumber() reject null on empty value, preventing invalid account data.
3. The method setInitialBalance() rejects negative values, ensuring financial correctness.
4. The Object's state can only be change through validation method, maintaing consistancy

IT23024

4. Write Program to -

i) Find the kth smallest element in an ArrayList

ii) Create a TreeMap to store the mappings of words to their frequency in a given text.

iii) Implement a Queue and stack using the priority Queue class with custom comparator

iv) Create a TreeMap to store the mappings of student IDs to their departments

Answer:

Array list of kth element

```
import java.util.*;

public class KthSmallest {

        public static void main(String[] args) {

        ArrayList<Integer> list = new ArrayList<>();

        List.add(7);
        list.add(2);
        list.add(9);
        list.add(1);

       int k=3;
```

```
        collection.sort(list);
        system.out.println("Kth smallest element:"+list.get(k-1));
    }
}
```

11) Tree Map word frequency count

```
import java.util.*;
public class WordFrequency{
    public static void main(String[] args){
        String text="Java is easy and java is powerful";
        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("word frequency:");
        for(Map.Entry<String, Integer> entry: map.entrySet(
            system.out.println(entry.getKey()+":"+entry.getV
        }
    }
}
```

IT23624

iii) Using priority queue stack and Queue implementat

```
import java.util.*;

class PQQueue{
    static class Element{
        int value, priority;
        Element(int value, int priority){
        this.value = value;
        this.priority = priority;
    }
}

static int count = 0;
static PriorityQueue<Element> pq = new priorityQueue<>(
    (a,b) -> a.priority - b.priority
);
static void enqueue(int x){
    pq.add(new element(x, count++));
}
```

```
static int dequeue(){
    return pq.poll().value;
}

public static void main(String[] args){

    enqueue(10);
    enqueue(20);
    enqueue(30);

    System.out.println(dequeue());
    System.out.println(dequeue());

    }
}
```

I123024

5. Developing a multithreading-based project to simulate a car parking management system with classes namely. RegistarParking - Represents a parking request made by a car; ParingPool - Acts as a shared synchronized queue; ParkingAgent Represents a thread that continously checks the pool and parks car from the queue and a MainClass - Simulates N cars arriving concurrently to request parking

car ABC123 requested parking.
Car XYZ456 requested Parking.
Agent1 parked car ABC123.
Agent2 parked car XYZ456.

...

Answer:

To simulate a car parking management system using multithreading, where multiple cars request parking concurrent and multiple parking agents process these requests from a shared synchroni queue.

IT23024

System Design

1. Registnan Parking
   i) Represents a panking request made by a can
   ii) stone a can numben.

2. Panking Pool
   i) Aets as a shaned synchnonized queque
   ii) Holds pending paning nequests
   iii) Ensune thnead safety

3. Panking Agent
   i) Repnesent a thnead
   ii) continously cheeks the pool
   iii) panks cans from the queue

4. Mainclass
   i) Simulates N cans anniving concunnently
   ii) stants multiple paning Agents

IT23024

1. RegistranParking class

```
public class RegistranParking {
        private string carNumber;
        public RegIstran Parking (strings carNumber) {
    this.carNumber = carNumber;
    }
public string getCarNumber (){
return carNumber;
}
}
```

2. ParkingPool class

```
import java.util.Linkedlist;
import java.util.Queue;

public class ParkingPool {
    private Queue< RegistarParking > Queue = new LinkedList<
// Add car to parking request queue
public synchronized void addRequest (Registranparking
request) {
queue.add(request);
System.out.println ("Car " + request.getCarNumber()+ "
  requested parking.");
notifyAll();
}
```

```
// Remove can from queue for panking
public synchronized RegisteranPanking getPeqod(),
    while (queue.isEmpty()){
     try{
        wait();
     } cateh (InterruptedException e) {
        e.printstackTnace();
       }
    }
    return queque.poll();
  }
}
```

3. panking Agent class

```
public class PankingAgent extends Thread {
      poivate PankingPool pool;
      poivate int agentId;

      public PankingAgent (PankingPool pool, int agentI
      this.pool = pool;
      this.agentId = AgentId;

   }
```

```
@Ovennide
public void nun(){
   while (true){
     RegistnanPanking  nequest = pool.getRequest();
    System.out.printIn( "Agent " + agentId + " panked can"
    + request.getCanNumben() + ", ")

   try {
    thread.sleep(1000); // simulate panking time
   } catch (IntennnuptedExceeption e){
        e.printStackTnee()'
      }
    }
  }
}
```

4. Mainclass

```
public class Mainclass{
      public static void main (stning[] args){
        panking Pool pool = new PankingPool();

        // start panking Agents
        PankingAgent agent1 = new PaningAgent (pool,1);
        PankingAgent agent2 = new PankingAgent(pool,2);
```

```
        agent1.start();
        agent2.start();

        // simulate cars arriving concurrently
        String[] cars = { "AB123", "XYZ456", "DEF789",
          "LMN321" };
        for (String car : cars) {
            new Thread(() -> {
             pool.addRequest(new RegistrantParing(car));
            }).start();
            }
    }
}
```

Sample Output:

```
Car ABC123 requeted parking.
Car XYZ456 requested parking.
Agent 1 parked car ABC123.
Agent 2 parked car XYZ456.
Car DEF789 requested parking.
Agent 1 parked car DEF789.
Car LMN321 requested parking.
Agent 2 parked car LMN321.
```

IT23024

6. How does Java handle XML data using DOM SAX parsers? Compare both approaches with respect to memory usage, processing speed, and use cases. provide a scenario when SAX would be preferred for DOM.

Answer:

Java Handles XML data using DOM and SAX parsers.

Java provide two main APIs to process XML data: DOM (Document object model) and SAX (Simple API for XML)

1. DOM parser

How it works:

i) The DOM parser reads the entire XML document and loads it in memory.

ii) It represent XML as tree structure

iii) The program can access, modify or traverse any node at any time.

Key characteristic:

i) Entire XML is stored in memory

ii) Support random access

iii) Allow modification of XML

Best suited for:

i) small to medium-sized XML files

ii) Application that need to update or traverse XML multiple times.

2. SAX parser

How it work

i) The SAX parser read the XML document sequentially.

ii) It is even-driven, triggering events like startElement, character, and endElement.

iii) It does not store the XML in memory.

key characteristics:

i) Reads XML as a stream

ii) Low memory usage.

iii) Read-only processing

Best suited for:

i) Very large XML file

ii) Fast processing when only specific data

IT23024

3. Comparison: DOM vs SAX

Feature	DOM parser	SAX parser
memory usage	High	Low
processing speed	slower	Faster
Access Type	Random access	Sequential Access
XML modification	possible	not possible
Ease of use	Easier	more complex
suitable XML size	small/medium	Large

4. Scenario where SAX is preferred over DOM

In banking or transaction processing system, a very large XML file containing daily transaction record must be processed, but only specific tag like <account Number> and <amount> are required.

IT23024

7. How does the virtual DOM in React improve performance? Compare it with the traditional DOM and explain the different algorithm with a simple component update example

Answer:

The virtual DOM is a lightweight JavaScript copy of real DOM. When state or props change, React updates the virtual DOM first, compare it with the previous version, and applies only the minimal required changes to the real DOM. This reduces expensive direct DOM manipulations.

Virtual DOM vs traditional DOM:

Aspect	Traditional DOM	virtual DOM
updates	Direct, immediate	Batched, optimized
performance	slower for frequent update	Faster
Re-rendering	Entire DOM may update	only changed parts update

IT23024

Diffing Algorithm:

React compares the old virtual DOM with the new one to find what changed. It then updates only those nodes in real DOM.

Example:

```
<h1>{count}</h1>
```

If count changes from 1 to 2, React detects that only the text value change and updates just that text node, not the whole page.

IT23024

8. What is event delegation in Javaseript and how does it optimize performance? Explain with an example of a click event on dynamically added element.

Answer:

Event delegation is a technique where you attach one event listener to a parent element instead of adding separate listeners to each child element. It work because of event bubbing, where events propagate from the target element up to its ancestors.

Improve performance.

i) Fewer event listeners - less memory usage

ii) Better performance for large or dynamic lists

iii) Automatic work for dynamically added elements

IT23024

Html

```
<ul id="list">
  <li>Item 1</li>
</ul>
```

Javacript

```
const list = document.getElementById("list");

list.addEventListener("click", function(e){
  if(e.target.tagName == "LI"){
    console.log("clicked:", e.target.textContent);
  }
});

// Dynamically adding element
const newItem = document.createElement("li");
newItem.textContent = "Item 2";
list.appendChild(newItem);
```

IT23024

9. Explain how Java Regular Expression can be used for input validation. write a regex pattern to validate an email address and desenibe how it work using pattern and matcher classes.

Answer:

Java Regular Expressions define a pattern that input data must follow. During input validation. the user input is matched against this pattern. If the input matches. it is considered valid, otherwise it is rejected. this is commonly used for validating emails phone number, passwords etc.

Email validation Regex Pattern

^[A-Za-z0-9+_.-]+@[A-Za-z0-9.+]+$

IC IT23024

Using pattern and Matcher classes.

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

public class EmailValidation{
        public static void main(String[] args){

            String email = "user@example.com";

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

        Pattern pattern = Pattern.compile(regex);

        Matcher matcher = pattern.matcher(email);

    if (matcher.matches()){

            System.out.println("valid email address");

    } else{

        System.out.println("Invalid email address");

            }

    }

}
```

IT23024

How it work

i) pattern.compile() converts the regex into a reusable pattern object.

ii) matcher() applies the pattern to the input string

iii) matches() checks whether the entire input matches the regex

Regex Explanation

i) ^ → start of string

ii) [A-Za-z0-9+_.-]+ → valid username characters

iii) @ → must contain @

iv) [A-Za-z0-9.-]+ → domain name

v) $ → end of string

IT23024

10. What are custom annotations in Java, and how can they be used to influence program behaviour at runtime using reflection? Design a simple custom annotation and show how it can be processed with annotation elements.

Answer:

Custom annotation in java:

i) User-defined metadata to add extra info to classes, methods, or fields.

ii) can influence runtime behaviour using Refection API

Code:

```
Import java.lang.annotation.*;
@ Retention ( RetentionPolicy.RUNTIME)
@ Target ( ElementType.METHOD)
@ interface MyAnnotation {
        String value();
}
```

IT23024

```
class d Demo{
    @MyAnnotation("Test")
    void show() { system.out.println("Hello"); }

}

public class Test {
    public static void main(String[] args) throws
Exception {
    method m = Demo.class.getMethod("show");
    if (m.isAnnotationPresent(MyAnnotation.class)){
    System.out.println(m.getAnnotation(MyAnnotation.class).v
    value());

    }
    }
}
```

1. @Retention(RUNTIME) → annotation avilable at runtim
2. Reflection checks annotated method & read s value
3. Program behaviour can change based on annotation.

IT23024

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

Answer:

Singleton Design Pattern

Ensure only one instance of a class exists and provides a global access point.

problem solved: Prevents multiple instance for resource like DB connections, logging, config, managers.

IT23024

How it work

1) private constructor - prevent external instantiation

11) static method (getInstance) - create instance once and return it

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

Thread safety

1. Synchronized method: ensures one thread creates instance
2. Double-checked locking: reduces synchroniz overhead.
3. static inner class: JVM ensures thread-s lazy intialization

IT23024

```
class Singleton{
    private Singleton(){}
    private static class Holden{ static final
    Singleton Instance = new Singleton(); }

    public static Singleton getInstance(){
    return Holden.Instance; }
}
```

12. Describe how JDBC manages communication between a Java application and a relational database. Outline the steps involved in executing a SELECT query and fetching results. Include error handing with try-catch and finally blocks.

Answer:

JDBC (java database connectivity) is a standard Java API that enables a Java Application to communicate with relational databases such as MySQL, Oracle, PostgreSQL etc. JDBC acts as a bridge between the java program and the database by using database-specific drivers.

JDBC follows a layered architecture:

1. Java Application
   sends SQL commands using JDBC API
2. JDBC API
   provides interfaces like Connection, Statement, ResultSet
3. JDBC driver
   converts Java JDBC calls into database-specific protocol
4. Relational Database.
   Executes SQL queries and return results

Steps to Execute a SELECT Query Using JDBC

step 1: Load and Register the JDBC Driver.

i) Loads the database driver into memory

ii) Enables Java to communicate with database

step 2: Establish a Database Connection

i) Uses DriverManager to connect to the database.

step 3: Create a Statement Object

i) Used to send SQL queries to the database

step 4: Execute the SELECT Query

i) Execute the query and returns a ResultSet Object.

step 5: Fetch Data from ResultSet

i) Reads record row by row

step 6: Close Resources

i) prevents memory leaks and resource exhaustion.

JDBC SELEct Query Example with Error Handing

```
import java.sql.*;
public class JDBCSelectExample {
    public static void main(String[] args) {
        Connection con = null;
        Statement stmt = null;
        ResultSet rs = null;
    try {
      // Step1: Load JDBC driver
      Class.forName("com.mysql.cj.jdbc.Driver");

    // Step2: Establish connection
      con = DriverManger.getConnection(
            "jdbc:mysql://localhost:3306/studentdb",
            "root"
            "mimi@170@M
      );

    // Step3: create statement
     stmt = con.createStatement();
```

```
// step4: Execute SELECT query
String query = "SELECT id, name, cgpa FROM students";
rs = stmt.executeQuery(query);

// step5: Fetch results
while (rs.next()) {
    int id = rs.getInt("id");
    String name = rs.getString("Name");
    double cgpa = rs.getDouble("cgpa");

    System.out.println(id + " " + name + " " + cgpa);

} catch (ClassNotFoundException e) {
    System.out.println("JDBC Driver not found.");
    e.printStackTrace();

} finally {
    // step6: close resource
    try {
        if (rs != null) rs.close();
        if (stmt != null) stmt.close();
        if (con != null) con.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}
```

Role of try-catch-finally in JDBC

try block

i) contains JDBC operations that may cause exceptions

ii) Ensure controlled execution

catch block

Handle:

i) class NotFoundException

ii) SQLException

finally block

i) Execute regardless of sucess or failure

ii) Ensure all database resources are properly closed

Advantage of Using JDBC

1. Platform independent
2. support multiple database
3. Secure and reliable data access
4. Standard API for database operations.

IT23024

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 Java as the model.

Answer:

Servlet : Handles client requests, interacts with the model, and forwards data to JSP.

JSP (view): Display data to the user.

Jave class : Encapsulation busines logic or data.

Code:

```
public class user{
        private string Name;
        public user (string name) {this.name=name;}
        public string getName() {return name;}
}
// controller (servlet)
@ WebServlet ("/greet")
public class UserServlet extends HttpServlet {
  protected void doGet
    User User = new User ("Alice");
    req.setAttribute ("user", user);
    req.getRequestDispatcher("view.jsp").forward
     (req, res);
}
```

14. Explain the life cycle of a Java servlet, where are the roles of the init(), service(), and destroy() methods? Discuss how servlets handle concurrent requests and how thread safety issues may rise

Answer:

Life Cycle Overview: A servlet goes through 3 main stages: initilization, request handing, and destruction.

1. init() – called once when the servlet is first loaded
2. service() – called for each client request
3. destroy() – called once before the servlet is unloaded.

Handing concurrent Request:

A single servlet instance handles multiple request via multiple threads

service() is called simultaneously in separate threads.

15. A single instance of a servlet handles multiple requests using threads. What problems can occur if shared resources are accessed by multiple threads? Illustrate your answer with an example and suggest a solution using synchronization.

Answer:

Problem:

A single servlet instance handle multiples request using multiple threads. If shared resource are accessed / modified by multiple threads simultaneously, it can lead to data inconsistency or race condition.

Problematic Code:

```
@WebServlet("/counter")
public class CounterServlet extends HttpServlet {
    private int count = 0;
    protected void doGet
        count++;
        res.getWriter().println("Count: " + count);
    }
```

Synchronization

```
@WebServlet("/counter")
public class CounterServlet extends HttpServlet {

    private int count = 0;

    protected synchronized void doGet

        count++;
        res.getWriter().println("Count: " + count);

    }

}
```

Alternative solution:

i) Use local variables inside doGet()

ii) Use AtomicInteger for atomic operations

16. Deserive how the MVC (Model-view-controller) pattern separates concerns in a Java web application. Explain the advantage of this structure in terms of maintainability and scalability, using a student register as an example.

Answer:

Separation of concerns:

1. Model: Encapsulates data and business logic
2. View: Handle presentation
3. Controller: Handles user request, interacts with the model, and forward data to the view.

Example:

model - student class stores student info, studentDAO handle database operation)

Controller: RegisterServlet process form input, calls studentDAO,

View: register.jsp displays the form. success.jsp show confirmation.

17. In a Java EE application, how does a servlet controller manage the flow between them model and the view. Provide a brief example that demonstrates forwarding data from a servlet to a JSP and rendering a response.

Answer:

In a Java EE (MVC) application, a servlet acts as the controller. Its main job is to:

1. Receive the client request. (HTTP request)
2. Invoke the model (business logic, service, DAO, entities)
3. store results in request / session scope
4. Forward the request to a view (JSP) for presentation.
   (The servlet does not generate HTML directly).

This keep business logic separate from UI, which is the core idea of MVC

Flow: Client → servlet (controllen) → Model → JSP(vie

Browsen
↓
servlet (controllen)
↓
Model (Business Logic/data)
↓
JSP (view)
↓
Response to browsen

1. servlet (controllen)

```
import jakarta.servlet.*;
import jakarta.servlet.http.*;
import jav.io.IOException;

public class studentServlet extends HttpServet{
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
                         throws ServletException, IOException{

        // model data (normally comes from service/DB)
        String studentName = "Mimi";
        int manks = 85;
        // stone data in request scope
        request.setAttribute("Name", studentName);
        request.setAttribute("manks", manks);
```

```
// Forward to JSP (view)
RequestDispatcher rd =
        request.getRequestDispatcher("result.jsp");

rd.forward(request, response);

    }
}
```

The servlet controls the flow
No HTML is written here
Data is passing using request.setAttribute()

2. JSP (view) - result.jsp

```
<html>
<head>
  <title> Result </title>
</head>
<body>
   <h2> student ResuH</h2>
   Name: ${name} <br>
   marks: ${marks} <br>
   <c:if test = "${marks> = 40}">
      <b> status: pass</b>
   </c:if>
</body>
</html>
```

1. JSP only ~~weld~~ handles presentation
2. Uses Expression language to access data
3. No business logic inside JSP.

18. Compare and contrast cookies, URL rewriting and HttpSession as methods for session tracking in servlets. Discuss their advantages, limitations and ideal use cases.

Answer:

1. Cookies:

i) How: Stores small data on the client browser
ii) Advantages: persistent across browser sessions
iii) Limitation: Disabled if client blocks cookies
iv) Use case: Remembering user preference login

2. URL Rewriting:

i) Appends session in URL
ii) Advantages: Works even if cookies are disabled
iii) URLs become messy, security risk
iv) Small application when cookies may be blocked.

3. Httpsession

i) Server-side storage of session objects each client gets a unique session ID.

ii) Secure, stores large objects, no dependency on client setting.

iii) Consumes server memory, expires often timeout.

iv) E-commerce. cart, login sessions, sensitive data

Comparison table:

Method	storage	client Dependency	Security	lifetime	Use case
Cookie	client	yes	medium	persistent or session	preferences, login
URL Rewriting	URL	NO	low	session only	small apps, no cookies
Httpsession	server	NO	high	session	sensitive data, cart

19. A web Application stores user login information using Httpsession. Explain how the session tracking in servelets. Process requests and how session timout on invalidation is handle securely

Answer:

Httpsession used in web applications to stores user login information and maintain user state across multiple request. Since Http is a stateless protocol, the server cannot remember user by default. Httpsession solves this problem.

How Httpsession is work

when a user log in, the server create an Httpsession and assigns a unique sessionID.

session timeout and Invalidation

for security a ssesion has a timeout period. If the user remains inactive for a certain time the session expire automatically and the user must log in again.

20. Explain how spring MVC handle an HTTP request from a browser. Describe the role of the @controller, @RequestMapping, and model objects in separating business logic from presentation. Provide a brief flow example of a login from submission

Answer:

spring MVC HTTP Request Handling

i) Spring MVC follows the Model-view-controller

ii) The browser sends an HTTP request to the dispatcherservlet

Role of key component

i) @Controller: Handles requests and connects business logic with the view

ii) @RequestMapping: Maps URLs and HTTP methods to controller methods.

iii) Model: Transfer data from controller to view.

21. Spring MVC uses the DispatcherServlet as a front controller. Describe its role in the request processing workflow. How does it interact with view resolvers and handler mapping.

Answer:

DispatcherServlet acts as the front controller in Spring MVC. It receives all HTTP requests, uses HandlerMapping to find the appropriate controller, and then forwards the request to it. After the controller returns a view name, dispatcherServlet uses a ViewResolver to resolve the actual view and sends the rendered response back to the client.

22. How does Prepared Statement improve performance and security over Statement in JDBC? Write a short example to insert a record into a MySQL table using Prepared Statement.

Answer:

prepared statement improves performance and security compared to statement in the following ways:

Performance Improvement

1. Precomplied SQL

   i) The SQL query is compiled once and reused

   ii) Faster execution when the same query runs multiple times.

2. Efficient Parameter Handling

   i) Parameters are passed separately, avoiding repeated SQL parsing.

Security improvement

1. Prevents SQL Injection

   i) User input is treated as data, not executable sc

   ii) Protects against malicious input like 'OR '1'='1

Statement limitation:

i) SQL is built using string concatenation
ii) Vulnerable to SQL injection.
iii) Slower for repeated queries.

Insert Record Using PreparedStatement (MySQL)

```
import java.sql.*;
public class IntentStudent{
    public static void main(String[] args){

    String url = "jdbc:mysql://localhost:3306/college";
    String user = "root";
    String password = "";
    String sql = "Insert INTO student (id, name, marks)
    VALUES (?, ?, ?)";

  try{
    // load driver
    Class.forName("com.mysql.cj.jdbc.Driver");

    // create connection
    Connection con = DriverManager.getConnection
     (url, user, password);
  // prepare statement
  PreparedStatement ps = con.prepareStatement(sql);
```

```
//set parameters
ps.setInt(1, 101);
ps.setString(2, "kanim");
ps.setInt(3, 88);

// Execute query
ps.executeUpdate();
System.out.println("Record inserted successfully");

// close resources
ps.close();
con.close();
} catch (Exception e){
        e.printStackTrace();
    }
}
}
```

Why Prepared statement is better

Feature	Statement	Prepared statement
SQL Injection Risk	High	None
Query Compilation	Every time	Once
performance	slower	Faster
Parameter support	No	Yes

23. What is resultSet in JDBC and how is it used to retrive data from a MySQL database? Briefly explain the use of next(), getString(), and getInt() method with an example.

Answer:

A resultset in JDBC is an object that holds the data returned by executing a SELECT query on a database. It represents a table of data where the cursor initially point before the first row.

The data is retrieved row by row using cursor movement method.

Commonly Used Resultset Methods

- next()
  moves the cursor to the next row
  Return true if a row exists, otherwise false.
  Must be called before accessing any data.
- getString(columnName / columnIndex)
  Retrives String data from the current row.
  Used for VARCHAR, CHAR, TEXT columns.
- getInt(columnName / columnIndex)
  Retrieves integer data from the current row
  Used for INT, NUMBER columns

Example: Retrieve Data from MySQL Using ResultSet

```
import java.sql.*;
public class FetchStudents {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/college";
        String user = "root"
        String password = "mimi@1701M";

        String sql= "SELECT id, name, marks FROM
            student";

    try{
      Class.forName("com.mysql.cj.jdbc.Driver");

     Connection con = DriverManager.getConnection(
                url, user, password);

     Statement stmt = con.createStatement();
     ResultSet rs = stmt.executeQuery(sql);

     // process ResultSet
```

```
while( rs.next() ){

    int id = rs.getInt ( "id" );    //getInt()

    String Name = rs.getString ( "name" ); //getString()
    int   marks = rs.getInt ( "marks" );

    System.out.println ( id + "  " + name + "  " + marks );

  }

 rs.close();
 stmt.close();
 con.close();

 } catch (Exception e) {
        e.printStackTrace();

     }

   }

}
```

ResultSet stores the result of a SELEct query

next() moves the cursor row by row

getString() retrives text data

getInt() retrives numeric data

24. How does JPA manage the mapping between Java objects and relational tables? Explain it with an Example using @Entity, @Id and @GeneratedValue annotations. Discuss the advantage of using JPA over raw JDBC

Answer:

JPA manage the mapping between Java objects and relational database tables using ORM. It follows developers to work with database data as Java object instead of writing SQL query manually

i) A Java class represent a database table
ii) class fields represent table comumn
iii) Annotation define how the mapping is done

```
@Entity
public class student{
        @Id
        @GeneratedValue
        private int id;
        private string name;
        private string email;
}
```

25. Describe the differences between the EntityManager's persist(), merge(), and remove() operations, when would you use each method in a typical database transaction scenario?

Answer:

persist:

i) Used to save a new entity to the database

ii) makes a new object persistent

iii) The entity becomes managed by the Entity manager.

iv) an Insert operation is performed.

merge():

i) Used to update an exsiting entity.

ii) copies data from a detached entity into a managed entity.

iii) Return a managed instance

iv) Perform an UPDATE

26. Design a simple CRUD application using Spring Boot and MySQL to manage student records. Desenibe how each operation (Create, Read, Update, Delete) would be implemented using a repository interface.

Answer:

A CRUD application perform four basic operations.

Create - Insert new data
Read - Retrieve existing data
Update - Modify existing data
Delete - Remove data

In Spring Boot, CRUD operations are easily implemented Using Spring Data JPA and a repository interface, with reduce boilerplate JDBC code

Overall structure

Client (Browser / postman)

↓

Controller (REST API)

↓

Service Layer (optional but recommended)

↓

Repository Interface (JpaRepository)

↓

MySQL Database

2. student Entity (Model)

```
import jakarta.persistence.*;

@Entity
@Table(name = "student")

public class student {

        @Id
        @GeneratedValue ( strategy = GenerationType.IDENT
        private Long id;

        private String Name;

        private int marks;

        // gettens and settens

}
```

i) Map directly to student table in MySQL
ii) @Id + @GeneratedValue handles primary key

3. Repository Interface

```
import org.springframework.data.jpa.repository.
JpaRepository;

public interface studentRepository
        extends JpaRepository<Student, Long>{

}
```

i) No method implementation needed
ii) Spring Data JPA automatically provides CRUD method

4. CRUD Operation Using Repository

CREATE - Insert a student

```
studentRepository.save(student);
```

i) If id is null, JPA perform INSERT
ii) Used for creating new student records.

READ - Fetch student data

Read all students

```
List < student > students = student Repository. findAll();
```

Read by ID

```
student student = studentRepository. findById (id) orElse
```

findAll() → SELECT *

findById() → SELECT where id = ?

Update - Modify Student Record

```
Student, student = studentRepository. findById (id), orElse
student. setMarks (90);
studentRepository. save (student);
```

i) if id exist → JPA perform Update

ii) Same save() method handles both create and update

DELETE - Remove student

```
studentRepository. deleteById (id);
```

i) Delete record using primary key

ii) Internally executes DELETE query

5. Controller example

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

    @Autowired
    private StudentRepository repository;

    @postMapping
    public student create(@requestBody Student s){

     return repository.save(s);
    }

  @getMapping
  public List<Student> getAll(){
     return repository.findAll();
  }
@putMapping("/{id}")
public student update @PathVariable Long id,
                      @RequestBody Student s){

    s.setId(id);
   return repository.save(s);
}
@DeleteMapping("/{id}")
  public void delete @pathvariable Long id){
      repository.deleteById(id);
}
```

6. Application properties

spring.datasource.url = jdbc:mysql://localhost:3306/college

spring.datastructure.username = root

spring.datasource.password =

spring.jpa.hibernate.ddl-auto = update

spring.jpa.show-sql = true

Operation	Repository Method
Create	save()
Read(All)	findAll()
Read (By ID)	findById()
Update	save()
Delete	deleteById()

remove()

i) Used to delete an entity from the database.

ii) the entity must be managed

iii) performs a DELETE operation

28. Compare @RestController and @Controller in Spring Boot. In a RESTful API for a library system, describe how you would structure the endpoint for books using REST principle.

Answer:

@Controller is used in Spring MVC to return views, while @RestController is used to build REST APIs and return data directly. In a Library REST API, book resource are exposed using endpoint like GET /book,

29. How does Maven manage dependencies and build lifecycle in a spring Boot project? Explain the structure of a typical pom.xml and how the Spring Boot starter dependency simplify development.

Answer:

Maven manages dependencies and the buid lifecycle in a Spring Boot project through the pom.xml file. The pom.xml defines project information, dependencies and build plugin. Spring Boot starter dependencies group commonly used libraries into a single dependency, reducing configuration and simplify development.

30. Demonstrate the project you developed with the important codes and Graphical User Interface.

Answer:

The project demonstrates how a Spring Boot Java application communicates with a relational database using JDBC. and displays the fetched data in a graphical web interface.

Backend: Spring Boot + JDBC

Fronted (GUI): HTML + thymeleaf

Database: MySQL

Operation: Execute SELECT query and show result in a table

Update Project Structure

```
springboot-jdbc-project
├── controller
│   └── StudentController.java
├── model
│   └── student.java
├── Repository
│   └── studentRepository.java
├── templates
│   └── students.html  <- GUI
├── Application.properties
└── SpringbootJdbcApplication.java
```

studentRepository.java

```
@Repository
public class StudentRepository {

    public List<Student> getAllStudents(){

    List<student> list = new ArrayList<>();

    try{
    Connection con = DriverManager.getConnection(
        "jdbc:mysql://localhost:330/studentdb",
        "root",
        "password"
       );

    statement stmt = con.createstatement();
    resultset rs = stmt.executeQuery(
        "SELECT id, name, cgpa FROM students"
       )
    ){
      while (rs.next()){
              list.add(new student(
                    rs.getInt("id"),
                    rs.getString("name"),
                    rs.getDouble("cgpa")

              ));
    } catch (SQLException e){
              e.printStackTrace();
       { return list;
    }
```

2. Controllen connection Backend to GUI

studentControllen.java

```
@ Controllen
public class studentControllen{
    private final studentRepositony repositty;
    public StudentControllen(StudentRepositony reposita
        this.repositony = repositony;
    }
@GetMapping("/students")
public string viewStudents(Model model){
    model.addAttnibute("students",
            repositony.getAllstudents());
    return "student";
    }
}
```

3. Graphical User Interface

student.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Student List</title>
    <style>
      body {
        font-family: Arial;
        background-colour: #f4f6f8;
      }
     table{
        bonder-collapse: collapse;
        width: 60%;
        margin: 50px auto;
        background: white;
     }
    th, td{
        padding: 12px;
        bonder: 1px solid #ccc;
       text-align: center;
    }
    th{
      background-colour: #2c3e50;
      colour: white;
    }
```

```
h2{
 text-align: center;
 margin-top: 40px;

 }
 </style>
</head>

<body>
<h2> Student Information (Spring Boot JDBC) </h2>

<table>
  <tr>
    <th> ID </th>
    <th> Name </th>
    <th> CGPA </th>

  </tr>
  <tr th:each = "s: ${students}">
        <td th:text = "${s.id}"></td>
        <td th:text = "${s.name}"></td>
        <td th.text = "${s.CGPA"></td>

    </tr>
  </table>
  </body>
  </html>
```

4. GUI output

http://localhost:8080/students

i) Clean web page
ii) student data displayed in table format
iii) Data fetch directly from database.

Example

ID	Name	CGPA
1	Mimi	3.75
2	Hritu	4.00

5. How the whole system works

1. User opens /students in browser
2. Controller receives request
3. Repository executes JDBC SELECT query
4. Data fetched using ResultSet
5. Data sent to Thymeleaf page
6. GUI displays record in table.

31. You are building a multi-module Spring Boot application with separate modules for API, service, and database. Explain how you would structure the project using maven and Gradle and inter module dependencies effectively

Answer:

In a multi-module Spring Boot application, the project is divided into separate modules such as API, service and database to achieve better separation of concern, reusability and maintainability

```
parent-project
├── pom.xml
├── Api-module
│   └── pom.xml
├── service-module
│   └── pom.xml
└── database-module
    └── pom.xml
```

33. socket programing example:

Answer:

Server Program:

```
import java.net.*;
import java.io.*;
public class Server{
    public static void main(String[] args) throws
    Exception{
    ServerSocket serversocket = new serversock
    Socket soeket = servensoeket.aceept()
    DataInputStream in =new DataInputStrem
    (soeket.getInputstream());
    String msg=in.nead UTF();
    System.out.printIn ("Client say:"+msg);
    servensoeket.close();
    }
}
```

Client program

```
import java.net.*;
import java.io.*;

public class Client{
    public static void main(String[] args) throws
        Exception{

    Socket socket = new Socket("localhost", 5000);

    DataOutputStream out = new DataOutput Stream
        (socket.getOutput stream());

    out.writeUTF("Hellow server");

    socket.close()
    }
}
```

34. RMI Example chat system with FX

Answer:

1. Remote Interface

```
import java.rmi.*;
public interface chatservice extends Remote{
    void sendMessage(String msg) throws
RemoteException;
{
```

2. JavaFX client

```
chatService chat =
(chatservice) Naming.lookup("rmi://localhost
                            /chat");
chat.sendmessage("Hellow from FX client
```

3. RMI server

```
Import java.rmi.*;
import java.rmi.server.*;

public class chatServer extends UnicastRemoteObject
implements chatservice{
        protected chatserver() throws RemoteExceptn
        {}

    public void sendMessage(string msg){
        System.out.println("client" + msg);

}
```