Meharin Choudhury

IT-23040

(1) Demonstrate how a child class can access a protected member of its parent class within the same package. Explain with example what happens when the child class is in a different package.

Ans:

protected access in same vs. different package.

Same package:

```
package p1;

class Parent {
    protected int x = 10;
}

class Child extends Parent {
    void show() {
        System.out.println(x);  // allowed.
    }
}
```

IT-23040

Different package:

```
package p2;
import p1.Parent;

class Child extends Parent {
    void show() {
        System.out.println(x);
        // allowed only via inheritance.
    }
}

class Other {
    void test(){
        Parent p = new Parent();
        // p.x not allowed.
    }
}
```

Rule:

protected is accessible:

- inside same package
- in subclass of another package (through inheritance, not object).

(2) Compare abstract classes and ~~inheritance~~ interfaces in term of multiple inheritance. When would you prefer to use an abstract class and when an interface?

Ans:

Abstract class vs. Interface:

Feature	Abstract class	Interface
Multiple inheritance	Not supported.	Supported
Methods	Abstract, concrete	Abstract (Default allowed)
Variables	Instance variables	public static final only
Constructors	Yes	No

Use abstract class when:

- Classes are closely related.
- You need state or, constructors.

Use Interface when:

- You need multiple inheritance.
- You define capability / contract.

IT-23010

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

Ans:

Encapsulation (BankAccount):

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

    public void setAccountNumber(String acc){
        if (acc == null || acc.isEmpty())
            throw new IllegalArguementException(
            "Invalid account number");
        this.accountNumber = acc;
    }
```

```
public void setInitialBalance(double amount) {
    if (amount < 0)
        throw new IllegalArguementException("Negative
          balance not allowed");
    this.balance = amount;
  }
}
```

Encapsulation ensures:

- Data security (private variables).
- Data integrity (validated setters).

(4) Write Program to - (Any 3).

(i) Find the kth smallest element in an ArrayList

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

(iii) Implement a Queue and stack using the

PriorityQueue class with a custom comparator.

(iv) Create a TreeMap to store the mappings of student IDs to their details.

(v) Check if two LinkedLists are equal.

(vi) Create a HashMap to store the mappings of employee IDs to their departments.

Ans:

(i) Kth smallest in ArrayList:

```
Collections.sort(list);
System.out.println(list.get(k-1));
```

(ii) Word frequency using TreeMap:

```
TreeMap<String, Integer> map = new TreeMap<>();
for (String w : text.split(" "))
    map.put(w, map.getOrDefault(w,0)+1);
```

(v) Check two LinkedLists equal:

list1.equals(list2);

(6) Developing a multithreading-based project to simulate a car parking management system with classes namely:

RegisterarParking - Represents a parking request made by a car;

ParkingPool - Acts as a shared synchronized queue;

ParkingAgent - Represents a thread that continuously checks the pool and parks cars from the queue and

MainClass - Simulates N cars arriving concurrently to request parking.

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

IT-23040

Agent 2 parked car XYZ456.

....

Ans:

Multithreaded Car Parking System:

```
class ParkingPool {
Queue<String> q = new LinkedList<>();
synchronized void request(String car){
   q.add(car);
   notify();
}
synchronized String park() throws InterruptedEx-
-ception {
   while(q.isEmpty()) wait();
   return q.poll();
  }
}
```

```
class ParkingAgent extends Thread {
    ParkingPool pool;
    ParkingAgent(ParkingPool p) { pool = p; }

    public void run() {
        try {
            while (true)
                System.out.println(Thread.currentThread().
                    getName() + "parked car" + pool.park());
        }
        catch (Exception e) {}
    }
}
```

(6) How does Java ~~had~~ handle XML data using DOM and SAX parsers? Compare both approaches with respect to memory usage, processing

speed, and use cases. Provide a scenario where SAX would be preferred over DOM.

Ans:

DOM vs SAX (XML):

Feature	DOM	SAX
Memory	High	Low
Speed	Slower	Fast
Access	Random	Sequential

Use SAX when XML is very large (log files, streaming data).

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

(7)

Ans:

Virtual DOM (React):

- Virtual DOM is a lightweight copy of real DOM.
- React updates only changed nodes using diffing.

jsx code:

```
setCount(count + 1);
```

Only the changed text node updates, not full page.

IT-23040

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

Ans:

Event Delegation (JS):

```
document.getElementById("list").addEventListener
    ("click", e => {
    if (e.target.tagName === "LI").alert(e.target.
    textContent);
});
```

Why: One listener handles many dynamic elements.

(9) Explain how Java Regular Expressions can be used for input validation. Write a regex pattern to validate an email address and describe how it works using the Pattern and Matcher classes.

Ans:

Java Regex (Email):

```
String regex = "^[\\w.-]+@[\\w.-]+\\.\\w+$";
Pattern p = Pattern.compile(regex);
Matcher m = p.matcher(email);
```

(10) What are custom annotations in Java and how can they be used to influence program behavior at runtime using reflection? Design a simple custom annotation and show how it

IT-23040

can be processed with annotated elements.

(10) Ans:

Custom Annotation + Reflection:

```
@Retention(RetentionPolicy.RUNTIME)
@Interface Info {
    String value();
}

@Info("Admin")
class User {}

Info info = User.class.getAnnotation(Info.class);
System.out.println(info.value());
```

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

Ans:

Singleton Pattern (Threade-safe):

```
class Singleton {
    private static volatile Singleton obj;
    private Singleton() {}
    public static Singleton getInstance() {
        if (obj == null) {
            synchronized (Singleton.class) {
                if (obj == null)
                    obj = new Singleton();
            }
        }
        return obj;
    }
}
```

IT-23040

(10) Describe how JDBC manages communication between a Java application and a relational database. Outline the steps involve in executing a SELECT query and fetching results. Include error handling with try-catch and finally blocks.

Ans:

JDBC SELECT Flow :

```
try {
    Connection con = DriverManager.getConnection(
      url, user, pass);
    Statement st = con.createStatement();
    ResultSet rs = st.executeQuery("SELECT * FROM
       student");

    while (rs.next())
        System.out.println(rs.getInt(1));
} catch (Exception e){
    e.printStackTrace();
```

```
} finally {
    con.close();
}
```

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

Ans:

Servlet + JSP (MVC):

- Servlet : Controller
- JSP : View
- Java class : Model

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

request.getRequestDispatcher("view.jsp").forward
(request, response);

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

Ans:

Servlet Life Cycle:

(1) init() → once

(2) service() → every request

(3) destroy() → cleanup.

Multiple threads call service().

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

Ans:

Thread Safety Issue:

```
int count; // shared

synchronized void increment() {
    count++;
}
```

IT-23040

(16) Describe how the MVC (Model-View-Controller) pattern separates concerns in a Java web application. Explain the advantages of this structure in terms of maintainability and scalability, using a student registration system as an example.

Ans:

- Model : Student
- View: JSP form
- Controller: Servlet

MVC Pattern Advantages:

- Separation of concerns
- Easy maintanace
- Scalable design.

Example:

Student registration system.

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

Ans:

```
request.setAttribute("name", name);
request.getRequestDispatcher("home.jsp").forward(
        request, response);
```

Servlet Controller flow:

- Receives request
- Interacts with model
- Forwards data to JSP using request dispatcher.

IT-22040

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

Ans:

Session Tracking Comparison:

Method	Pros	Cons
Cookie	Simple	Insecure
URL rewriting	No cookies	Ugly URLs
HttpSession	Secure	Server Memory

Cookies:

- Client-side
- Less secure

URL rewriting:

- Adds session ID to URL
- Not user-friendly.

HttpSession:

- Server-side.

- Most Secure.

(10) A web application stores user login information using HttpSession. Explain how the session works across multiple requests and how session timeout or, invalidation is handled securely.

Ans:

HttpSession Working:

```
HttpSession session = request.getSession();
session.setAttribute("user", user);
session.invalidate();
```

HttpSession Management:

- Session created on first request
- Session ID shared across request.
- Timeout or, invalidate() ends session.

IT-23040

(20) Explain how Spring MVC handles 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 form submission.

Ans:

Spring MVC Login Flow:

- @Controller receives request
- @RequestMapping maps URL
- Model sends data 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 mappings?

(21) Ans:

DispatcherServlet:

- Front Controller in Spring MVC
- Calls handler mapping
- Resolves view.
- Receives all requests.

(22) How does PreparedStatement improves performance and security over Statement in JDBC? Write a short example to insert a record into a MySQL table using PreparedStatement.

Ans:

```
PreparedStatement ps = con.prepareStatement("INSERT
    INTO emp VALUES (?, ?)");
ps.setInt(1,1);
ps.setString(2, "HR");
ps.executeUpdate();
```

IT-203040

Prevents SQL Injection.

PreparedStatement:

- Faster
- Prevents SQL injection.
- Precompiled SQL.

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

Ans:

ResultSet:

```
while (rs.next()) {
    rs.getInt("id");
    rs.getString("name");
}
```

ResultSet in JDBC:

- Holds database query results
- next() moves cursor
- getInt(), getString() retrieve values.

(Q4) How does JPA manage the mapping between Java objects and relational tables? Explain with an example using @Entity, @Id, and @GeneratedValue annotations. Discuss the advantages of using JPA over raw JDBC.

Ans:

JPA Mapping:

```
@Entity
class Student {
    @Id
    @GeneratedValue
    int id;
}
```

IT-23040

- @Entity maps class to table
- @Id marks primary key
- @GeneratedValue auto-generates ID.

Advantage:
Less code than JDBC.

Q5) Describe the difference between the EntityManager's persist(), merge(), and remove() operations. When would you use each method in a typical database transaction scenario?

Ans:

- persist() → new object (insert)
- merge() → update detached (update)
- remove() → delete (delete)

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

Ans:

CRUD Spring Boot (Student):

```
interface StudentRepo extends JpaRepository <
    Student, Integer> {}
```

- Repository interface extends JpaRepository
- Handles Create, Read, Update, Delete automatically.

IT-23040

(Q7) How does Spring Boot simply the development of RESTful services? Describe how to implement a REST controller using @RestController, @GetMapping and @PostMapping including JSON data handling.

Ans:

```
@RestController
class StudentController {
    @GetMapping ("/students")
    List<Student> getAll () {}
}
```

- @RestController → returns JSON
- @GetMapping → fetch data
- @PostMapping → insert data.

(28) Compare @RestController and @Controller in Spring Boot. ~~and MySQL~~ In a RESTful API for a library system, describe how you would structure the endpoints for books using REST principles (e.g., /books, /books/{id}).

Ans: REST endpoints:

- GET /books
- GET /books/{id}
- POST /books

- @Controller → JSP/HTML
- @RestController → JSON.

(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 dependencies simplify development.

IT-23040

(29)

Ans:

Maven (pom.xml)

- Dependencies (manages)
- Build life cycle (handles)
- Spring Boot starters reduce configuration.

(30) Compare Maven and Gradle as build tools in Spring Boot projects. Discuss their syntax, performance and dependency handling with an example of a build.gradle configuration for a simple REST API.

Ans: Maven vs Gradle:

Feature	Maven	Gradle
Syntax	XML	Groovy
Speed	Slower	Faster

IT-23040

gradle:

implementation 'org.springframework.boot:spring-boot-starter-web'