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 API's 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 procesing when only specific data is needed

I123024

3. Comparison: DOM vs SAX

Feature	DOM parsers	SAX parsers
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 Javascript 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>
```

Javascript

```
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 email 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 character

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.printIn("Hello"); }

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

1. @Retention(RUNTIME) → annotation avilable at runtime
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 instance 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, manager.

How it work

1) private constructor - prevent external instantion

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 synchronization overhead.
3. static inner class: JVM ensures thread-safe 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 databasees 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 Queny Example with Error Handing

```
impont java.sql.*;
public class JDBCSeleet Example {
    public static void main (string[] args) {
        connection con = null;
        statement stmt = null;
        Resultset rs = null;
    try {
      // step1: Load JDBC drivers
      class.forName("com.mysql.cj.jdbc.Driven");

    // step2: Establish connection
      con = Driven Manger.getConnection(
            "jbdbc:mysql://localhost:3306/studentdb";
            "root"
            "mimi@1170@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 ~~is~~ independent
2. Support multiple database
3. Secure and reliable data access
4. Standard API for database operations.