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.

Event delegation is a technique where a single event listener is added to a parent element, and it handles events for its child element using event bubbling.

Event delegation reduces/optimizes the performance by reducing the number of event listeners needed in a web page. Instead of adding individual listeners to each child element, a single listener is attached to a parent element. This take advantages of event budding, where events on child elements propagate up to their parents. When an event occurs on a child, parent's listener handles it, effectively managing event for all descendants.

Example of click event on dynamically added elements:

Context:

You have an empty <ul> list. Users can click a button to add new <li> items. When any <li> is clicked, you want to show an alert with the item's text. If you use event deligation, you add just one listener

to the <ul> but if you use no event deligation then, youll need to attach a click listener to every new <li>.

9. Explain how java regular expressions can be used for input validation. Write a regrex pattern to validate an email address and describe how it worcks using the Pattern and Matcher classes.

Java Regular expressions are now a powerful toolkit for input validation, allowing you to define patterns that strings must match to be considered valid. They're particularly useful for validating formats like email-address, phone numbers etc.

Java provides regex functionality through the java.util.regex package, primarily using two classes:

Pattern: Represented a compiled regular expression.
Matcher: Performs matching operations on a character sequence using a pattern.

Regex pattern for a basic email:

^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6}$

Here,

^ → start of the string

[a-zA-Z0-9._%+-]+ → Local part

@ → must include @ symbol

[a-zA-Z0-9.-]+ → Domain name

\. → Dot (.)

[a-zA-Z]{2,6} → Top level domain (com, org)

$ → End of the string

How the regex pattern works using Pattern and Matcher Class is given below:

Pattern Class:

- Pattern.compile(regex) - Compiles regex into a pattern object.
- Pattern.matches(regex, input) → static method for one time matching

Matcher Class:

- matcher.matches() → Tests if entire string matches the pattern.
- matcher.find() → Finds the next occurrence of the pattern
- matcher.reset() → Resets the matcher to reuse it.

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 annotated elements.

Custom annotations are user defined metadata that can be attached to code elements. When combined with reflection they let your program →

i) Detect annotations at runtime
ii) Read their values
iii) Change behaviour dynamically

Step by step process of how custom annotation works is given below →

i) Define a custom annotation using @interface.
ii) Use @Retenation to make it accessible at runtime.
iii) Attach the annotation to classes, methods or fields.
iv) Use Java Reflection API to read and process the annotations.
v) Apply logic/behaviour based on the annotation value.

How custom annotation influence program behaviour at runtime using Reflection is given below:

1. Define a custom annotation

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

2. Use the Annotation in a Class

```
public class MyTask{
  @RunNow(times = 3)
  public void printHello(){
    System.out.println("Hello");}
  public void notAnnotated(){
    System.out.println("This won't be called");}
}
```

3. Process it with reflection at Runtime

```
import java.lang.reflect.Method;
public class AnnotationRunner{
  public static void main(String[] args) throws Exception{
    MyTask task = new MyTask();
  for(Method method: task.getClass().getDeclaredMethods()){
```

```
if (method.isAnnotationPresent(RunNow.class)){
RunNow annotation= method.getAnnotation(RunNow.
                                        class);
  int times = annotation.times();
for (int i=0; i<times; i++)
    method.invoke(task);
} } } }
```

Output:

```
Hello
Hello
Hello
```

11. Discuss the singleton design pattern in java. What problem what 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

The Singleton design Pattern is one of the most well known creational patterns that ensures a Class has only one instance throughout the application lifecycle while providing global access to that instance.

The Singleton Pattern addresses several key issues:

i) Preventing multiple instances

ii) Only one object is needed to coordinate actions across the system.

iii) Resource management.

The pattern uses several mechanism to guarantee that only one instance is created. They are -

i) Private constructor

ii) Static Factory Method

iii) Handling reflection attacks.

iv) Serialization safety

v)

To achieve thread safety in a Singleton implementation their are some methods:

1) Synchronized method

Code:

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

```
            instance = new Singleton();
        }
        return instance;
    }
}
```

2. Bill Pugh Singleton

Code:

```
public class Singleton{
    prive singleton(){}
private static class Holder{
    private static final Singleton INSTANCE=new
                                        Singleton();
}
public static Singleton getInstance(){
    return Holder.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 handling with try-catch and finally blocks.

JDBC is an API that allows Java applications with to interact with relational databases like MySQL, PostgreSQL and Oracle etc.

How JDBC manages communication:

JDBC acts as a bridge between Java and the database using:

i) DriverManager - loads the JDBC driver.
ii) Connection - estabishes the session
iii) Statement - sends SQL commands
iv) ResultSet - holds the data returned from queries.

Here's a concise outline of the steps to execute a SELECT query using JDBC with error handling:

1. Load the JDBC Driver
2. Establish a connection
3. Prepare SQL query.

4. Set parameters
5. Execute the query
6. Process the Resultset
7. Handle Exceptions
8. Closure resource in finally block.

Example of catch and finally block:

```
try {
      //set parameters, execute
   }
catch (SQLException e) {
      //handle sql errors
    }
finally {
      //close resultset, statement
   }
```

13. How do servlets and JSPs work together in a web application following the MVC architecture? Provide a brief use case showing the servlet as a controller, JSP as a view and a Java class as the model.

In a Java web application using MVC architecture, servlets, JSPs and Java classes each play distinct roles to cleanly seperate logic, presentation and data. How they work together is given below:

1. Client sends an HTTP request
2. Servlet receives the request
3. Extracts data from the request
4. Forward result to JSP for display
5. Model performs data processing
6. JSP (view) displays the result.

Use case:

Java class as model

UserValidator.java

```
public class UserValidator{
    public static boolean isValid(String username, String
                                  password){
        return username.equals("admin") && password.
                                  equals("1234");
    }
}
```

View using JSP

result.jsp

```
<%
Boolean isValid = (Boolean)request.getAttribute("isValid");
if(isValid != null && isValid) {
%>
  <h2> Welcome, admin! </h2>
<% } else {
  <h2> login failed. Try again. </h2>
<% } %>
```

Controller using servlet

LoginServlet.java

```
@WebServlet("/login")
public class LoginServlet extends HttpServlet {
protected void doPost(HttpServletRequest request,
        HttpServlet Response response)
throws ServletException, IOException{
String user = request.getParameter("username");
String pass = request.getParameter("password");
boolean isValid = UserValidator.isValid (user, pass);
request.setAttribute("isValid", isValid);
RequestDispatcher rd = request.getRequestDispatcher
                        ("result.jsp");
rd.forward(request, response); }
}
```

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

The servlet life cycle defines the stages a servlet goes through from its creation to destruction in a servlet. The Javax.servlet package provides interfaces for handling these stages.

Roles of methods:

init()

→ Called once after the servlet is instantiated.
→ Used to initialize resources.
→ The ServletConfig object provides access to initialization parameters.

service()

→ Called every time the servlet receives a request.
→ It dispatches to doGet(), doPost() etc, depending on request method.

destroy()

→ Called once before the servlet is taken out of service.
→ Clean up resources like closing DB connections.

How servlet handles concurrent Requests is given below and also why thread safety matters -

Servlet Request Handling

- The servlet container (Tomcat) creats one instance of each servlet class.
- For every incoming request, the container spwans a new thread and invokes the servlets service method -
- This means many threads can be executing the servlet at the same time.

Thread safety concerns

Since multiple threads share the same servlet instance, any shared or mutable instance variables can be accessed by multiple threads concurrently. This can lead to -

- Race conditions
- Inconsistent results
- Unexpected behaviour.

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

When multiple threads access shared resources then the given problems arises:

In a servlet one instance serves many requests simultaneously. The servlet container spawns a new thread for each request.

If instance variables are read or modified by these threads without coordination, problems arise - such as:

i) Race conditions → threads interface with each other unpredictably.

ii) Data inconsistency → variables can have unexpected values.

iii) Incorrect results → one user's data may leak into another user's response.

An example problem using race condition and its solution using synchronization is given below:

Example:

```
private int counter=0;
counter++;
response.getWriter().println("Request number:"+counter
```

In this problem 2 threads run nearly at the same time.

Solution:

To prevent this problem, we can use synchronization. Synchronization means access to the shared variable so only one thread at a time can execute that criticalsection.

```
synchronization(this){
 counter++;
response.getWriter().println("Request number:"+counter);
```

Here the synchronization –

- Locks the servlet instance before entering the block.
- Esures only one thread updates counter at a time.

16. Describe how the MVC patterern seperates 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.

The MVC pattern is a widely used design pattern in Java web applications, especially with spring MVC, Struts and JSF etc frameworks. It seperates concerns by dividing the application into three main components each responsible for a specific aspect of the application.

1) Model → Business logic and data

i) Responsibility: Manages the application data, rules & logic

ii) Examples: Java classes that represent data, service classes and DAO.

iii) Concern seperated: Business logic is isolated from UI and request handling.

2) View:

i) Responsibility: Displays data to the user and handles user interface.

ii) Example: JSP, Thymeleaf etc.

iii) Concern seperated: UI rendering is kept seperate from data and control logic.

3) Controller:

i) Responsibility: Acts as an intermediary between View and Model.

ii) Example: Spring @Controller classes.

iii) Concern seperated: Input processing and routing are seperated from data logic and presentation.

Using student registration system, the advantages of MVC pattern interms of maintainability and scalability is given below:

Maintainability:

- If there is a bug in data storage, you only check the Model.
- UI designers can work on JSP without touching business logic.

Scalability:

- You can plug in different views using the same controller and Model.
- New features (e.g. 'view student list') can be added by just introducing new components without breaking existing ones.

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

In a Java EE application, a servlet controller manages the flow between the model and the view by:

1. Receiving requests from the client
2. Calling model classes to process data or fetch from the database.
3. setting attributes on the request scope.

Flow overview:

Client → Servlet → Model → JSP(view) → client.
(controller)

Example: Student Info

1. Model Class

```
Public class Student{
    private String name;
    private int age;
Public Student (String name, int age){
    this.name = name;
    this.age = age;  }
```

```
public string getName(){ return name;}
public int getAge(){
                    return age;}
```

2. JSP(View):

```
<%@ page imports="your.package.student" %>
<%
 Student student=(Student)request.getAttribute("student
                                                Data");
%>
<html>
 <head><title>Student Info</title></head>
<body>
   <h2>Student details</h2>
   <p> Name: <%= Student.getName() %></p>
   <p> Age: <%= Student.getAge() %></p>
</body>
</html>
```

3. Controller(Servlet):

```
@WebServlet("/student")
public class StudentServlet extends HttpServlet{
protected void doGet(HttpServletRequest request, Http
                                    ServletResponse response)
throws ServletException, IOException{
 Student student= new Student("Afita", 22);
 request.setAttribute("studentData",student);
```

```
RequestDispatcher dispatcher = request.getRequestDispatcher
                    ("student.jsp");
dispatcher.forward(request, response); } }
```

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

The comparison table is given below:

Feature	Cookies	URL rewriting	HttpSession
Mechanism	Store session id or data in browser cookies.	Appends session id to the URL as a parameter.	Stores data on server side with session ID in client
Storage Location	Client	Client	Server
Persistence	Can persist beyond browser session	only active during session	Exists for session duration
Automatic Handling	Yes	No	Yes
Security	Medium	Low	High
Data Capacity	Limited	Limited	High

Advantages, limitations and ideal use cases of each methods is given below:

Cookies

- Small piece of data stored on the client side.
- Sent with each HTTP request.
- Used to track user info.

Advantages:

- Persistent across sessions.
- Easy to implement.

Limitations:

- Limited in size.
- Security risks.

Ideal use case:

- Remembering user preferences.

URL rewritting

- Session ID is manually added to every URL as parameter.
- Useful when cookies are disabled.

Advantages

- No reliance on browser cookies.
- Simple for all small apps.

Limitations:

- Security risk
- Bookmarked URL's leaked session info.

Ideal use case:

Public web apps where cookies can't be used.

HTTPSession

Advantages

- Secure
- Scalable
- Automatic management by servlet container

Limitations

- Requires server memory
- Data lost if server restarts.

Ideal use case

- Authenticated user sessions, carts, profiles etc.

19. A web application stores user login info using HttpSession. Explain how the session works across multiple requests and how session timeout or invalidation is handled securely.

In a Java web application, HttpSession is a powerful mechanism that allows you to track and manage user specific data across multiple HTTP requests - like login state, cart contents or user preferences.

How HTTPSession Works Across Requests:

i) ~~How~~ User Logs IN!

- A new session is created
- The server generates a unique Session ID.
- This ID is usually sent to the browser via a cookie.

ii) Subsequent Requests:

- The browser automatically sends the JSESSIONID cookie.
- The server uses this ID to retrive the existing session object.

Session Timeout and Invalidation

- **Session Timeout**

1. A session will expire automatically after a period of inactivity.
2. This is configured in web.xml.
3. After the timeout, the session object is destroyed, and the session ID becomes valid.

- **Invalidation**

1. You can explicitly invalidate a session using →
   request.getSession().invalidate();

Secure Handling of Session

1. Use HttpOnly and Secure Cookies
2. Use HTTPs
3. Regenerate Session on Login
4. Validate Session Attributes
5. Short Timeout for sensitive Data.

20. Explain how Spring MVC handles an HTTP request from a browser. Describe the role of the @Controller, @RequestMapping and Model objects in seperating business logic from presentation. Provide a brief flow example of a login form submission.

In spring MVC, handling an HTTP request involves a well structured flow that follows the Model-View-Controller pattern. This architecture seperates business logic, request handling, and presentation making the application modular, maintainable and testable.

Spring MVC Request Handling Flow

When a browser spends a request -

1. The DispatcherServlet receives the request
2. It looks for a controller method mapped to the request URL.
3. The matched @Controller method process the request.
4. It uses a model object to add attributes for the view.

5. The method returns a viewname and the DispatcherServlet renders it.

The role of business logic in seperating business logic is given below:

1. @Controller →
   - Keeps the business logic in services
   - Connects request routing to backend processing.

2. @RequestMapping →
   - Directs with controller method responds to a given URL.
   - Allows clean Routing seperate from both business logic and UI rendering.

3. Model Object →
   - Avoids embedding data processing logic in views.
   - Keeps UI and backend logic loosely coupled.

Example of login form submission –

```
@controller
public class LoginController{
    @PostMapping ("/login")
    public String handleLogin(
        @RequestParam String username;
        @RequestParam String password.
```

```
    Model model) {
if ("admin".equals(username) && "pass".equals(Password))
    model.addAttribute("user", username);
    return "loginSuccess"; }
else {
    model.addAttribute("error", "Invalid credentials");
    return "loginform"; }
} }
```

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

Dispatcher servlet Interaction Diagram:

Browser Request

↓

DispatcherServlet

↓

HandlerMapping → Controller Method

↓

Model → View Name

↓

ViewResolver

↓

Render view with model

↓

Browser receives HTML response

Role of Dispatcherservlet in Request Processing:

1. Receives request:

- All requests mapped to spring MVC or are first received by Dispatcherservlet.

2. Delegates to HandlerMappings:

- The servlet consults HandlerMapping beans to determine which controller method should handle the request.

3. Invokes the controller

- The identified controller method is invoked with any parameters.

4. Processes Return Value

- The controller method returns a logical view with data.

5. Resolves the view:

- Dispatcherservlet uses a ViewResolver to convert the logical view name to an actual view.

6. Renders the view.

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

In JDBC, a ResultSet is an object that holds the data returned from executing a select query on a relational data such as: MYSQL. It represents a table of data. Each row in the result set can be accessed one at a time using a cursor that move forward.

How it retrives data is given below:

(. rs.next() moves the first row.)

- Establish a database connection
- create a statement
- Execute the query
- Iterate through the ResultSet
- Retrive column values.
- Process the retrived data.
- Close Resources.

Use of common methods with example:

- next(): Moves the cursor to the next row. Returns false when there are no more rows.
- getString(): Retrives the values of the specified column as a string.
- getInt(): Retrives the value of the specified column as an int.

Example:

```
String query = "select id, name, marks from students";
ResultSet rs = stmt.executeQuery(query);
while (rs.next()){
  int id = rs.getInt("id");
  String name = rs.getString("name");
  int marks = rs.getInt("marks");
System.out.println(id + " | " + name + " | " + marks);
}
```

23. How does JPA manage the mapping between Java objects and relational tables? Explain with an example using @Entity, @Id and @Generated Value annotations. Discuss the advantage of using JPA over raw JDBC.

JPA manage the mapping between Java objects and relational databases tables through a process called Object-Relational Mapping.

Here's how JPA handle the mapping:

Entity definition

- Java classes are marked as JPA entities using the @Entity annotation. This indicates that the class represents a table in the database

Field to column Mapping

- By default, JPA maps fields in an entity class to columns in the corresponding database table with the same time.

Relationship mapping

JPA provides annotations to manage relationships between entities, which correspond to relationships between tables in the database.

Example, with @Entity, @Id and @GeneratedValue annotation:

```
import jakarta.persistance.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.GeneratedValue;
@Entity
public class Student{
  @Id
    private int id;
    private String name;
public int getId(){
            return id; }
public void setId(int id){
          this.id = id; }
public String getName(){
          return name; }
public void setName(String name){
          this.name = name; }
}
```

The advantages of using JPA over raw JDBC for database interaction Java applications -

i) Object - Relational Mapping

ii) Reduced Boilerplate Code

iii) Database Independence

iv) Improved maintainability and Readability

v) Enhanced Productivity

24. Design a simple CRUD application using Spring B and MySQL to manage student records. Describ how each operation would be implemented using a repository interface.

Project overview:

i) Add a student (create)

ii) Get all or single student (read)

iii) Update a student's data (update)

iv) Delete a student (delete)

① Database Table

```
create database studentdb;
use studentdb;
```

② Entity Class - Student.java

```
import jakarta.persistence;
@Entity
public class student;
```

```
@Id
@GeneratedValue
  private long id;
  private string name;
  ----- - -
  --- - - -

}
```

3. Repository Interface

```
import org.springframework.data.jpa.repository.
                                   JpaRepository;
public interface StudentRepository extends JpaRe-
            pository<Student, Long> {
.....
}
```

This interface gives built in methods like:

- save()
- findById(), findAll()
- deleteById()

4. Controller class

```
import org.springframework.web.bind.annotation;
import java.util.List;
@RestController
@RequestMapping(" /students")
```

```
Public class StudentController{
  private final StudentService service;
Public StudentController(StudentService service)
    this.service = service;
}
    //create
@PostMapping
public student addStudent(@RequestBody Student st
{
 return service.createStudent(student);
}
// add
// update
// delete
}
```

25. How does spring Boot simplify the development of RESTful services? Describe how to implement a REST controller using @RESTController, @GetMapping and @PostMapping. including JSON data handling..

spring Boot greatly simplifies the development of Restful services using :

i) Auto configuration
- Automatically configures web servers and JSON converters with minimal setup.

ii) Embeded server:
- No need to deploy WARs to an external server. Just run your app like a Java application.

iii) Spring web starter
- Includes all required dependencies for buiding REST API's.

iv) Reduced Boilerplate
- Annotations like @RESTController, @GetMapping, @PostMapping etc. simplify request handling.

Impletmentation of REST controller using annotations and JSON data handling :

1)@REST Controller Annotation:

- Marks the class as a REST API controller.

- It combines @Controller and @ResponseBody

2) @GetMapping

- @GetMapping is used to map HTTP Get requests
- Typically used to retrive data from server.

3) @POSTMapping

- @POSTMapping maps HTTP post requests.
- Used to send data to server.
- Conversion here is handled using @RequestBody

④ JSON data handling

- The JSON is sent by the client
- Spring converts this JSON into a Java object using Jackson.
- This is done by making the method parameter with @RequestBody.

26. How does PreparedStatement improve performance and security over statement in JDBC? Write a short example to insert a record into a MySQL table using PreparedStatement.

How PreparedStatement improves Performance & security over statement in JDBC is given below:

Security:

- PreparedStatement automatically escapes user input making it safe from SQL injection attacks.
- With Statement, user input is directly concatenated into the SQL string.

Performance:

- PreparedStatement allows the database to pre-compile the SQL statement and reuse it with diff. parameters.
- This reduces parsing time and improves efficiency.

Example

Inserting a Record Using PreparedStatement

```
import java.sql.*;
public class InsertExample{
```

```
public static void main(String[] args){
 String url="jdbc:mysql://localhost:3306/studentdb"
 String username="maisha";
 String pass = "root-1236";
String insertSQL="insert into students(name,age)
                  values(?,?)";
try(Connection conn=DriverManager.getConnection(
    url,username, password);
 pstmt.setString(1,"Afifa");
 pstmt.setInt(2, 22);
int rowInserted = pstmt.executedUpdate();
  if(rowsInserted>0)
    sout("Successful insertion");
}
 catch(SQLException e){
    e.printStackTrace();
} }
  }
```

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

Difference table is given below:

Method	Purpose	Return value	When to use
persist()	Makes a new entity managed and scheduled it for insertion	void	When inserting a new record
merge()	copies the state of a detached entity into a managed entity	Managed entity	When updating an entity
remove()	Delets a managed entity from the database	void	When deleting a record.