ABU SAYEM
IT-22008

TOPIC NAME : DAY
TIME DATE : / /

02/ Comparison between abstract classes & interfaces in terms of multiple inheritance:-

Feature	Abstract class	Interface
multiple inheritance	not supported	Fully supported.
Extends	class A extends class B	class A implements x, y, z.
Code reuse	can have method bodies and member variable	Java 8+ can have default and static method.
state	can have instance variables	Only constants (public static final)
Construct-or.	Yes. (can initialize field)	No constructor allowed.

When to use an abstract class:

(i) To provide base functionality and shared states.

GOOD LUCK

TOPIC NAME : DAY : TIME DATE / /

(ii) When need constructor or non-static instance variables.

(iii) When expected closely related classes with an "is-a" relationship

o When to use interface :-

(i) Want to define pure behaviour, not implementation.

(ii) Classes are unrelated but share common capabilities.

(iii) Want to use multiple inheritance of type.

3.) How does encapsulation ensure data security and integrity? show with an bank account class using private variables and validated methods such as SetAccountNumber(String) etc.

Answer :- Encapsulation is a key principle in object-oriented programming that hide internal data. It helps data

GOOD LUCK

CamScanner

TOPIC NAME : DAY : TIME : DATE : / /

security, data integrity, maintainability.

```
public class BankAccount{
private String accountNumber;
private double balance;
public void setAccountNumber(String
                                AccountNumber){
if(accountNumber==null)|| accountNumber.
                                trim().isEmpty()}
{
 throw new IllegalArgumentException("can't be
                                    null"), }
 this.accountNumber = accountNumber; }
public void setInitialBalance(double balance){
if(balance<0){
throw new IllegalArgumentException("Balance
                                can't be negative)
}
 this.balance = balance; }
public String getAccountNumber(){
return AccountNumber; }
public double getBalance(){
  return balance; }
```

GOOD LUCK™

TOPIC NAME : DAY : TIME DATE : / /

```
public void deposit (double amount){
if (amount > 0) this. balance += amount;
}
}
```

14. Find k^{th} smallest element.

(1)

```
import java.util.* ;
public class KthSmallest{
public static void main (String [] ags){
List <Integer> list = Arrays. aslist (8,2,5,1,9,4);
Collection. sort (list);
int K = 3;
System. out. println(K+ 'rd smallest', list
                                get(K-1));
}}
```

Output : 4

GOOD LUCK

CamScanner

TOPIC NAME : DAY TIME DATE / /

(ii) Word frequency using treemap

```
import java.util.*;
public class WordFreq {
public static void main(String[] ags) {
String text = "apple banana apple mango";
TreeMap<String, Integer> map = new TreeMap
for (String word: text.split(" ")) <> ();
map.put(word, map.getOrDefault(word, 0)+1);
map.forEach((k,v) -> System.out.print(k+"
                                    ="+v)); } }
```

Output:

apple = 2

banana = 1

mango = 1

(iii) Queue & stack using Priority Queue.

```
import java.util.*;
public class PQStackQueue {
 Static class Element {
 int val, order;
Element(int v, int o) { val=v; order=o; }}
```

TOPIC NAME

```
public static void main(String [ ] args){
priority Queue<Element> stack = new
  priority Queue<>((a,b)->b.order- 0.order);
Priority Queue<Element> queue = new Priority
Queue<>(comparator.comparing]n] (a->a.order
  int order=0;
stack.add(new Element(10, order++));
Stack.add(new Element (20, order++));
System.out.println("Stack pop:"+ stack.poll().
                                  val))
queue.add(new Element (100,0));
queue.add(new Element(200.1));
System.out.println("Queue poll:"+ queue.poll().
                                  val)}}
```

7. Developing a multithreading-based project to simulate a car parking management System.

multithread based project

```
import java.util.*;
import java.util.concurrent.*;
class ParkingPool{
private final Queue<String> queue=
                    new linkedlist<>();
```

TOPIC NAME : DAY TIME : DATE : / /

```
public synchronized void addCar(String car) {
queue.add(car);
System.out.println("Car"+car+" requested parking.");
notify(); }
public synchronized String getCar() {
while (queue.isEmpty()) {
try {wait(); } catch (InterruptedException e)
{ }
return queue.poll();
}
}
class RegisterParking extends Thread {
Private final String CarNumber;
private final ~~String~~ ParkingPool pool;
Public RegisterParking(String carNumber, ParkingPool pool) {
this.CarNumber = CarNumber;
this.pool = pool; }
Public void run() {
    pool.addCar(CarNumber); }}
```

GOOD LUCK

CamScanner

TOPIC NAME : DAY : TIME : DATE : / /

```
Class ParkingAgent extends Thread{
Private final ParkingPool pool;
Private final int agentId;
Public ParkingAgent(ParkingPool.pool, int agent.
                                        Id){
this.pool = pool;
this. agent Id = agentId; }
public void run() {
  while(true) {
String car = Pool. getCar();
System. out. Println("Agent" + agentId + " Parked
                                car + car + ". ");
try {Thread. sleep(500);} catch(InterruptedException e)
                                        {}
}
}
}public Class CarParkingSystem{
public static void main(String[] arg){
ParkingPool pool = new parking Pool();
New parkingAgent(Pool, 1). start();
new Parking Agent (Pool, 2). start();
new RegistrarParking("ABC123", pool). start();
new RegistrarParking("XYZ456", pool). start();
```

TOPIC NAME : DAY TIME DATE / /

6. Comparison between DOM vs SAX

Feature	DOM	SAX
memory	High (loads whole xml)	Low (reads line by line
speed	slower for big files	faster for large file
Navigation	Easy (tree structure)	Hard
modification	yes	no
Best for	small xml, Editing	Large xml

7. How does the virtual DOM in React improve performance?

React creates a virtual copy of DOM. On update. React:

1. Compares (diffs) old and new Virtual DOM.
2. finds whats changed
3. Applies only the changes to real DOM.

GOOD LUCK

CamScanner

TOPIC NAME : DAY : TIME : DATE : / /

18. Event delegation in Java Script

A technique where a single event listener is attached to a parent element to handles events from current and future child elements

```
<ul id='menu'>
<li> Home </li>
<li> About </li>
</ul>
document.getElementById("menu").addEventListener(
"click", function(e){
if (e.target.tagName=="LI"){
alert("Clicked:" + e.target.innerText);
}});
```

09. 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.

GOOD LUCK

CamScanner

TOPIC NAME : DAY TIME DATE / /

Java provides the java.util.regex package which contains Pattern and Matcher classes to work with Regular Expressions (regex). These can be used for input validation like email addresses, phone numbers, passwords etc.

1. Pattern: A compiled representation of a regex.
2. Matcher: Used to perform match operations on a character sequence using a Pattern.

- Steps for Input Validation using Regex in Java:

1. Define a regex pattern
2. Compile the pattern using Pattern.compile().
3. Create a matcher using Pattern.matcher(input String).
4. Use matcher.matches() to check if the input matches the pattern.

- Regex Pattern to Validate Email Address:

^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,6}$

GOOD LUCK

CamScanner

TOPIC NAME : DAY : TIME : DATE : / /

□ Java Example Code to Validate Email Address:

```
import java.util.regex.Matcher;
import java.util.regex.patterns
public class EmailValidator {
 public static void main(String[] args) {
String email= "emam.email123@gmail.com";
String regex= "^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]
                    +\\.[A-Za-z]{2,6}$";
Pattern pattern = Pattern.compile(regex);
Matcher matcher = Pattern.matcher(email);
if (matcher.matches()) {
System.out.println(email+ "is a valid email
                                  address.");
} else {
        System.out.println(email+ "is an invalid
                                  email.address.");
 }
}
}
```

TOPIC NAME

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 in java are user-defined annotations created using the @interface keyword. They can be used to add metadata to program elements. By themselves, annotations do not affect program behaviour, but using Reflection API, you can inspect annotations at runtime and modify program behaviour accordingly.

o Define a Custom Annotation

```
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
import java.lang.annotation.ElementType;
public @interface MyAnnotation {
  String value() default "Default value";
}
```

TOPIC NAME : DAY : TIME : DATE : / /

□ process Annotation using Reflection

```
import java.lang.reflect.Method;
public class AnnotationProcessor {
  public static void main(String [] args)
throw Exception{
  MyService service = new MyService();
Method[] methods = service.getClass().getDeclaredMethods();
for (Method method: methods) {
if (method.isAnnotationPresent(MyAnnotation.class)){
MyAnnotation annotation = method.getAnnotation(MyAnnotation.class);
System.out.println("Found annotation with value: " + annotation.value());
method.invoke(service);
}
}
}
```

1 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.

The singleton Design pattern ensures that only one instance of a class is created throughout the lifetime of an application and provides a global access point to that instance.

- problems It Solves :-

(i) Control object creation to avoid multiple instances of a class where only one shared instance makes sense.

(ii) Useful when exactly one object is needed to coordinate actions across the system.

Thread-Safe Singleton Implementations

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

TOPIC NAME : DAY : TIME : DATE : / /

2. Describe how JDBC manage 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.

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

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

- SELECT query using JDBC with error handling:

1. Load the JDBC Driver.
2. Establish a connection.

3. Prepare SQL query.
4. Set parameters, Excute the query
5. Process the Result set
6. Handle Exceptions
7. closure resource in finally block.

Example of catch and finally block:

```
try {
    // set parameters, execute
}
catch (SQL Exception 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 clearly 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 request
4. Forward result to JSP for display
5. Model performs data processing
6. JSP (view) display the result.

<u>Use Case</u>: Java class as model

```
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");
```

TOPIC NAME : DAY TIME DATE / /

```
if (is valid != null && is valid) {
%>
<h2> Welcome, admin! </h2>
<% } else {
<h2> Login failed. Try again </h2>
<% } %>
```

[14] Explain the life cycle of Java servlet. What are the roles of init(), service(), destroy(), methods? Discuss how servlets handle concurrent request 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 intialization parameters.

. service()
→ called every time the servlet receives a request.

TOPIC NAME : DAY : TIME : DATE : / /

→ It dispatches to do Get(), doPost() etc. depending request method.

- destroy ()

→ called once before the servlet is taken out of service.

→ clean up resource like closing DB connection

- Servlet Request Handling

→ The servlet container (Tomcat) creates one instance of each servlet class.

→ 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

GOOD LUCK

CamScanner

TOPIC NAME DAY TIME DATE / /

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

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

(i) Race conditions → threads interface with each other unpredicatbly

(ii) Data inconsistency → Variables can have unexpected values.

(iii) Incorrect results → one user's data may leak into another usersers's response;

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.

TOPIC NAME : DAY : TIME : DATE : / /

means access to the shared variable so only one thread at a time can executed that critical section.

```
Synchronization(this){
    counter++;
response.getWriter().Println("Request number:"+
                                counter)
```

Here the synchronization

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

16) Describe how the MVC pattern seperates concerns in a Java Web application. Explain the advantages of this structure in terms of maintainability using a student registration system as an example.

The MVC pattern is a widely used design pattern in java web application especially with spring MVC, Struts and JSF etc frameworks. It seperates concerns

TOPIC NAME : DAY : TIME DATE : / /

by dividing the application into three main components each responsible for aspecific aspect of the application.

1. Model → Business logic and data
Manages the application data, rules. Java classes that represent data service classes and DAO.

2. View
Displays data to the user and user interface like JSP, Thymeleaf etc.

3. Controller
~~Respon~~ Acts as an intermediary between View and Model. Spring Controller classes.

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

GOOD LUCK

CamScanner

TOPIC NAME

DAY

PAGE

DATE / /

Scalability :

You can plug in different views using the same controller and Model. New features can be added by just introducing new components without breaking existing ones.

7. In a Java EE application, how does a servlet controller manage the flow between the model and the view? Provide a brief ~~example~~ 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 ~~to~~ attributes on the request scope.

TOPIC NAME DAY TIME DATE / /

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 get Name(). { return name; }
public int get Age(){ return age;}
```

2. JSP View

```
<%@page import, 'your. package. Student" %>
<html>
<head><title> Student Info</title></head>
<body>
    <h2> Student Details </h2>
    </body>
    </html>
```

3. Controller

```
@web servlet("/Student")
public class StudentServlet extends HttpServlet
throws servlet Exception. IOException{
    student student= new student( 'Sayem', 22);
dispatcher. forward(request. response); }}
```

TOPIC NAME : DAY : TIME : DATE : / /

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

The comparison table is given below:

Feature	Cookies	URL rewriting	Http Session
Mechanism	Stores session id or data in browser cookies	Appends session id to the URL as a parameter	stores data on server side with session id on client
Storage location	client	client	server
Automatic Handling	yes	No	yes
Security	Medium	Low	High
Data capacity	Limited	Limited	High
persistence	can persist beyond browser session	only active during session	Exists for session duration

TOPIC NAME : DAY
TIME DATE / /

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

Cookies:

- Persistent across session
- Easy to implement
- limited in size
- security risk
- Remembering user performances.

URL rewriting:

- No reliance on browser cookies
- Simple for all small apps
- security risk
- public web apps where cookies can't be used.

HTTP session:

- secure
- Scalable
- Automatic management by servlet container
- Requires server memory
- Data lost if server restart
- Authenticated user sessions carts, profiles.

GOOD LUCK

CamScanner

TOPIC NAME : DAY : TIME : DATE : / /

19 A web Application stores user login info using HttPsession. Explain how the session works across multiple requests and how session time out or invalidation is handled securely.

In a Java web Application, HttPsession is 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 works:

(i) User Login

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

(ii) Subsequent Requests:

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

TOPIC NAME : DAY : TIME : DATE : / /

Session Time out

1. A session will expire automatically after period of inactivity.
2. This is configured in web.xml.

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 Attribute
5. Short time out 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 brief flow example of a Login form submission.

GOOD LUCK

TOPIC NAME : DAY : TIME : DATE : / /

In MVC, handling an HTTP request involve Spring a well structured flow that follows the Model → View → controller patterns.

- Request Handling flow —

1. The Dispatcher Servlet receives the request.
2. It looks for a controller method mapped to the request URL.
3. It use a model object to add attributes for the view.
4. The method returns a viewname and the Dispatcher servlet renders it.

1. @ controller →
- Keeps the business logic in servlets.
- connects request routing to backend processing

2. @ Request Mapping →
- Directs with controller method responds to a given URL

3. Model Object →
Avoids embedding data processing logic in views.

TOPIC NAME DAY TIME DATE / /

[21] Spring MVC uses the Dispatcher Servlet as a front controller. Describe its role in request processing workflow. How does it interact with resolver and handler mapping?

The workflow of Spring MVC as front controller—

1. Request Received: The browsers sends an HTTP request. Dispatcher servlet intercepts the request.

2. Handler Mapping: Consults Handler Mapping to determine which controller method should handle request.

3. Controller execution: It invokes method controller method. The method process the business logic and returns to logical view name

4. View Resolution: Delegates to a View Resolver

5. Process rendering: Spring boot MVC renders the view (HTTP, JSON, etc) and send the HTTP response back to client.

GOOD LUCK

CamScanner

TOPIC NAME : DAY : TIME : DATE / /

22. How does prepared statement improve the performance and security over statement in JDBC? Write short example to insert a record into a My SQL Table using prepared statement?

Prepared statement improves performance and security in the following way -

1. Prepared statement pre-complies the SQL query. So repeated excutions are faster.
2. It supports batch() and excute Batch() for bulk insertion
3. It uses parametrized queries (?) so user input is treated as data not excutable SQL
4. Special characters (', ") are escaped safely

Code: import java.sql.*;

```
public class InsertExample {
public static void main(String[] args){
String URL = "jdbc:mysql://localhost:3306/mydb";
String user = "Sayem";
```

```
String password = "1.2.34";
try(connected conn = DriverManager.get Connection
                    (URL, user, password);
prepared Statement pstmt = conn.
    prepared Statement(SQL) {
  pstmt set String (1, "Sayem");
    pstmt. set String(2, 'atsf54@gmail.com');
int rows Affected = pstmt.executed
                    update();
System. out. println (rows Affected + " rows
                    inserted.");
catch (SQL Exceptione) {
    e. printstack Trace ();
}
}
}
```

TOPIC NAME : DAY : TIME : DATE : / /

(24.) JPA (Java Persistance API) uses annotations to map java classes to database tables. It abstracts SQL and JDBC, letting you work with objects.

Key Annotations:

① @Entity → Marks a class as a JPA entity (mapped to a table)

② @Id → Defines primary key.

③ @ Generated value → specifies how the primary key is generated.

Code:

```
import jakarta.persistance.*;
@Entity
    public class Student {
@Id
@ Generated value ( strategy = GenerationType. IDENTITY)
private long id;
private String Name;
private int age;
```

GOOD LUCK

TOPIC NAME : DAY : TIME : DATE : / /

"Gettens and settens
}

Here the student class is mapped to a student table. Field → colums.

o Advantage of JPA over Raw JDBC.

→ JPA is Automatic and JDBC manual.

→ Transactions in JPA ~~But~~ Built in but in Raw JDBC manual.

→ JPA maintain ability High. Raw JDBC is low.

→ JPA ~~object mapping~~ code length short and clean. Raw JDBC is verbose.

GOOD LUCK

CamScanner

TOPIC NAME : DAY : TIME : DATE : / /

25

Method	Purpose	Use When
persist()	Insert new entity	Adding a new object to the database
merge ()	Update or reattach	Updating a database or existing entity
remove()	Delete entity	Deleting a managed entity

Transaction Scenario Example :

```
Entity transaction tx = entityManager
  get Transaction ();
tx. begin ();
Student newStudent = new Student
        ("Tom", 21);
entityManager.persist (newStudent);
student existing = entityManager.
```

GOOD LUCK

TOPIC NAME : DAY
TIME : DATE : / /

```
find(Student.class, 51);
existing.setAge(25);
entityManager.merge(existing);
entityManager.remove(existing);
tx.commit();
```

GOOD LUCK

CamScanner

TOPIC NAME : DAY
TIME : DATE : / /

30.

Feature	Maven	Gradle
Syntax	XML(pom.xml)	Groovy (kotlin) built. gradle
performance	Slower	Faster
Dependency.	Stable, declara-tive	Flexible, scriptable
Readability	Verbose	concise

• Build.gradle Example :

```
groovy
plugins {
  id 'org.springframework.boot'
    version '3.2.5'
id 'io.spring.dependency-
  management' version '1.1.5'
```

GOOD LUCK

TOPIC NAME : DAY : TIME : DATE : / /

```
id 'java'
}
group = 'com.example'
version = '1.0.0'
sourceCompatibility = '17'
repositories {
 maven central ID }
dependencies {
implementation 'org.springframework.
boot:spring-boot-starter-web'
test Implementation 'org.springframework.boot:spring-boot-
    starter-test'
}.
```

GOOD LUCK