Lab - 1 : Compare abstract classes and interfaces in terms of multiple inheritance. When would you prefer to use an abstract class and when an interface?

⇒ Abstract class :-

1. Supports single inheritence only
2. Contain abstract methods
3. Can contain concrete methods
4. Can contain instance variable and constructors.

Limitation : Do not support multiple inheritence.

Interface :

1. Supports multiple inheritance
2. Contains abstract methods
3. Contains constants
4. Also contains default and static methods.
5. No constructors or instance variables

Advantage: Allows a class to inherit behaviour from multiple sources, enabling flexible design.

When to use abstract class :-

i) when class are closely related

ii) When you need to share code, state or constructor

iii) When partial implementation is required.

When to use interface :-

i) When multiple inheritance is needed.

ii) When defining capabilities or roles.

iii) When classes are unrelated but share common behaviour.

iv) When loose coupling is required.

Lab-2 : 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.

=> Encapsulation and Data Security :-

Encapsulation restricts direct access to an objects data by :

1. Declaring variables as private
2. Allowing control access through public methods.

This prevents unauthorized modification ensures that only valid data is stored.

How it ensures integrity :

i) Invalid values are rejected

ii) Business rules are enforced at a point.

iii) Objects remains in a consistent secure state.

Eample : BankAccount class

```
public class BankAccount{
        private string accountNumber;
        private double balance;
public void setAccountNumber (String accountNumber){
    if (accountNumber == null || accountNumber.trim().isEmpty){
      throw new IllegalArgumentException("Invalid acc");
    }
 this.accountNumber = accountNumber;
}
 public void setInitialBalance (double balance){
        if (balance <0) {
    throw new IllegalException ("Balance can not negative");
    }
    this.balance = balance;
}
public string getAccountNumber (){
        return accountNumber;
 }
public double getBalance (){
        return Balance;
    }
 }
```

Lab-3 :- System Description : The system simulates multiple cars arriving simultaneously and requesting parking. A shared parking pool manages these requests, while parking agents process and park the cars safely.

Class Description :

1) RegisterParking : Represents a parking request made by a car.

2) ParkingPool : A shared, synchronized queue that stores parking requests.

3) ParkingAgent : A thread that continuously retrives and parks cars from the pool.

4) MainClass ; simulates multiple cars arriving concurrently.

#Implementation

```
import java.util.LinkedList;
import java.util.Queue;
```

```
class RegistranParking {
    private String carNumber;
    public RegistranParking (String carNumber) {
        this.carNumber = carNumber;
    }
    public String getCarNumber()}
        return carNumber;
    }
}
class ParkingPool {
 private Queue < RegistranParking > queue = new LinkedList();
 public synchronized void addCar (RegitranParking car) {
        queue.add (car);
        notify ();
    }
public synchronized RegistranParking getcar () throw{
            InterruptedExcep tion {
    while (queue.is Empty ()){
        wait();
      }
    return queue.poll();
    }
}
```

```
class ParkingAgent extends Thread {
    private ParkingPool pool;
    public ParkingAgent (ParkingPool pool) {
        this.pool = pool;
    }
    public void run () {
    try {
     while (true) {
        RegistranParking car = pool.getCar ();
        System.out.println ("Parking car :" + car.getNumber ());
       Thread.sleep (500);
        }
     }
      catch (InterruptedException e) {
      System.out.println ("Parking Agent stopped");
     }
    }
  }

public class MainClass {
        public static void main (String[] args) {
        ParkingPool pool = new ParkingPool ();
        ParkingAgent agent = new ParkingAgent (Pool);
         agent.start ();
        for (int i=1; i<=5; i++) {
          pool.addCar (new RegistranParking ("CAR-" + i));
        }
      }
    }
```

Lab-4 :

JDBC Overview : JDBC is an API that allows java applications to connect to a relational databases, execute SQL queries and retrive and process results.

Steps to Execute SELECT Query :

i) Load the JDBC driver

ii) Establish a database connection

iii) Create a statement.

iv) Execute the SELECT query

v) Process the result set

vi) Close resources.

Example :

```
import java.sql.*
public class JbdcExample {
    public static void main (String [] args){
    Connection conn = null;
    statement stmd = null;
    ResultSet rs = null;
try {
    class.forName("com.mysql.cj.jdbc.Driver");
    conn = DriverManager.getConnection("Jdbc:mysql
```

```
://localhost:3306/testdb","user","password");
stmt = conn.createStatement();
rs = stmt.executeQuery("SELECT * FROM students");
while (rs.next()) {
 System.out.println(rs.getInt("id")+" "+rs.getString
                                        ("name"));
   }
 }
 catch (Exception e) {
   System.out.println("Database error"+e.getMessage());
 } finally {
    try {
       if (rs!=null) rs.close();
       if (stmt!=null) stmt.close();
       if (conn!=null) conn.close();
     }
      catch (SQL Exception e) {
      system.out.println("Error closing resources");
      }
    }
   }
}
```

Lab: 5 :- Role of Servelet Controllen :

In MVC architech-ture -

→ Model handles business logic and data.

→ View (JSP) renders the user interface

→ Controllen (servlet) :

1. Receive client requests.
2. Interacts with the model.
3. Forwards data to the view

This separation improves maintainability and scalibility.

Example: Forwarding data from servlet to JSP :

```
import java.IOException;
import javax.servlet.*;
import javax.servlet.http.*;

public class studentServlet extends HttpsServlet{
    protected void doGet(HttpServletRequest req,
                         HttpServletres ponseres)
    throws ServletException, IOException{
     String studentName = ("Alice");
     req.setAttribute("name; studentName);
```

```
RequestDispatcher rd = req.getRequestDispatcher(
                        "student.jsp");
    rd.forward(req,res);
  }
}
```

JSP (view)

```
<html>
<body>
  <h2> Welcome, {name} </h2>
</body>
</html>
```

Lab - 6:

Prepared Statement Advantages:

Performance:

i) SQL query is precompiled once and reused.

ii) Faster execution for reapeated queries.

Security:

i) Prevents SQL injection

ii) Automatically handles input sanitization.

Comparison:

Statement	Prepared Statement
1. Query compiled every time.	1. Query compiled once.
2. Vulnerable to SQL injection	2. Secure against SQL injection
3. String concatenation is needed.	3. Use placeholders (?) instead string concatenation.

Example : Insert Record Using Prepared Statement :

```
import java.sql.*;
public class insertExample{
    public static void main (String[] args){
    try{
     Connection con = DriverManger.getConnection(
        "jdbc:mysql://localhost:3306/testdb","user","pass");
 String sql = INSERT INTO student (id, name)
                    VALUES (?,?)";
   PreparedStatement ps = con.prepareStatement(sql);
   ps.setInt(1,101);
   ps.setString(2,"John");
   ps.executeUpdate();
    con.close();
 }
   catch (SQLException e){
     System.out.println("Database error:"+getmessage());
   }
  }
}
```

Lab - 1 :

ResultSet Overview : A ResultSet is an object that stores the data returned by executing a SELECT query in JDBC. It represents a table of data and allows you to iterate through the query results.

Commonly Used Methods :

1. next() : Moves the cursor to the next row in the result set.
2. getString() : Retrieves column data as a string
3. getInt() : Retrieves column data as an integer

Example : Retrieving Data from MySQL

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

```
Connection con = DriverManager.getConnection(
"jdbc:mysql://localhost:3306/test-db","user","password");
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT
        id, name, FROM students");
while (rs.next()){
    int id = rs.getInt("id");
    String name = rs.getstring("name");
    System.out.println(id + " " + name);
  }
  con.close();
 }
  catch(SQLExceptionc){
  System.out.println("Database error" +
 }                          e.getMessage());
 }
}
```

Lab-8:

How Spring Boot Simplifies REST Development:

Sprint Boot:

1) Eliminates complex configuration
2) Provides embeded servers like Tomcat
3) Automatically converts java objects into JSON using Jackson.
4) Uses annotations for quick REST API development.

REST Controllers Annotations:

i. @RestController : combines @Controller and @ResponseBody

ii. @GetMapping : Handles HTTP GET requests

iii. @PostMapping : Handles HTTP Post requests

iv. @RequestBody : Maps JSON data to java objects.

Example : REST controllen with JSON Handling :

```
import org.springframework.web.bind.annotation.*
@RestController
@RequestMapping("/api")
public class StudentController{
    @GetMapping("/Student")
  public student getstudent(){
    return new student(){
   }
 @postMapping("/student")
  public string addStudent(@RequestBody Student studn())
   return "Student added"+ student.getName();
 }
}
class student{
        private int id;
        private String name;
    public Student(int id, String name){
        this.id = id;
        this.name = name;
    }
  public int getId() {return id;}
  public String getName(){ return name;}
}
```

Lab-9:

Import code Snippets:

1. Import and window setup:

```
from tkinter import *
root = Tk()
root.title("Simple Calculator")
root.geometry("320x420")
root.resizable(False, False)
```

2. Display and Global Expression:

```
expression = ""
display = StringVar()
display_field = Entry(root, font='arial', 18),
                bd=10, insertwidth=4, width=14,
                borderwidth=4)

display field.grid(row=0, colum=0, columnspan=4,
                padx=10, pady=10)
```

3. Button click functions

```
def press (num):
    global expression
    expression = str(num)
    display.set(expression)

def equal():
    global expression
    try:
        result = str(eval(expression))
        display.set(result)
        expression = result
    except:
        display.set("Error")
        expression = ""

def clear():
    global expression
    expression = ""
    display("")
```