Lab Report-1:

Lab title: Comparison Between Abstract class and interfaces in Terms of Multiple Inheritence in Java.

Theory:

Multiple Inheritence: Multiple inheritence refers to a feature where a class can inherit properties and behaviors from more than one parent class. Java does not support multiple inheritence using class to avoid ambiguity. but the supports multiple inheritence using interface.

Abstract class:

An abstract class is a class that cannot be instiated and may contain.

- abstract methods (without body)
- Non-abstract methods (with body)
- Instance variables
- constructors

Lab Report : 01

Lab Title : Comparison between Abstract Class and Interfaces in Terms of Multiple Inheritance in Java.

Theory :

Multiple Inheritance : Multiple inheritance refers to a feature where a class can inherit properties and behaviors from more than one parent class. Java does not support multiple inheritance using classes to avoid ambiguity, but it supports multiple inheritance using interfaces.

Abstract class :

An abstract class is a class that cannot be instantiated and may contain:

- Abstract methods (without body)
- Non-abstract methods (with body)
- Instance variables
- Constructors

Lab Report - 2

Encapsolation is an OOp concept that hides data and protect them from direct access. It ensures data security by making variable private inside as class. Outside class cannot change data directly. Data can be changed only using public methods. These method check the values before saving them. This prevents wrong or harmful data from entering the system.

Encapsolution also keep data consistent and correct. If invalid data is given, the method reject it. In bank system, this is a very important for safety.

Therefore, encapsolution ensure both data and security.

Java code:

```
Class Account {
    privet string account number;
    private double balance;
    public void set Account Number (string accNo){
    if (accNo != null && ! accNo.is empty ()){
      account Number = accNo;
    }
    ebe {
   system.out.print ln ("Invalid account Number");
    }
  }
 public void set Initial Balance (double amount){
 if (amount >= 0){
 balance amount;
 }
 else {
 System.out.println ("Balance cannot be neg.");
 }
}
}
```

```
While (rs.next()){
  system.out.println (rs.getInt(1)+" "+rs.getString(2));
 }
}

 catch (Exception e) {
  system.out.println ("Error occured");
 }
    finally {
    try {
      if (con!=null)
        con.close ();
      }
      catch (Exception e) {
    system.out.println ("connection not closed");
  }
 }
}
}
```

Lab Report 2.9

JDBC stands for Java Database Connectivity.

It is used for connect a (data) java program with a relational database. JDBC work as a bridge between java application and database. The java program sent SQL queries using JDBC. JDBC driver receives the request and talks to the database. The database process the query and sends to the result back. JDBC driver receives/returns the result to the java program. JDBC allows data to be inserted, update delete and read. It also handles connection and error management. Thus, JDBC manages smooth connection between java and Database. To execute a select query JDBC follows some fixed steps.

Then a statement prepared statement object is created. After that, a SELECT SQL query is written. The query is executed using executeQuery method. This method return a Result set. try block is used to handle database operation safely. catch block handle SQL or runtime errors. Finally block is used to close connection and resourness.

Java code:

```
import Java.sql.*;
class select Example {
  public static void main (String [] args) {
    Collection con = null;
try {
      con = DriverManager.getConnection (
"Jdbc:mysql://localhost:3306/testdb", "root",
    statement st = con.creatStatement ();
```

Lab Report-5

In a Java EE application, a servlet works as a controller. The controller manages the flow between model and view. The model contains between business data and logic. The servlet gets data from the model. Then it sends this data to the view. JSP used as the view to show data to the user. The servlet forword the request to Jsp using request attribute JSP reads the data and displays it. Thus.

Java code:

```
import Java.io *;
import Javax servlet.*;
import Javax.http.*;
public class Hellow servlet extends Http servlet {
protected void doGet (Http servletrequest req.
        Http servlet Response res.
```

```
throws servlet Exception ,IO Exception {
 String name = " Student ";
 req.set Attribute ( " msg " ; name);
Request dispatcher rd = req.get Request Dispatcher
                              " hellow.jsp ");
      rd.forward (req, res);
   }
}
```

Jsp code for view:

```
<html>
<body>
 <h2> Hellow. ${msg} </h> }
</body>
</html>
```

Lab Report - 6.

Prepaid statement is used to excute SQL queries safely in JDBC. It improves performan because the qury is precompiled by the database. The same query can be used many time with different values. Prepared statement is faster than statement for repeired quieres. It also improves security by preventing SQL injection attacks. User input is treated as data, not as SQL code. Statement directly excutes SQL and is less secure. Prepared statement uses placeholder (?) for values. These values are set using setter methods so prepared - statement is better for the performance and security.

Java Code (Insert using pregared statement):

```
import java.sql.*;
class insertExcute {
  public static void main (String [] aarge)}
try {
connection con = Driver Manager.get connection (
" Idbc: my sql; // localbost : 3306 / destdb; "root"; "pass
word ");
string sql = "Insort Into student (id, name) values (?)";
prepared statement ps = con. prepared statement
                                    (sql)
  Ps.setInt (1,1);
  Ps. set string (2,'Jeamin");
  Ps . execute update ();
 System.out. println ("Record inserted");
 con.close ();
 }
 catch (Exception) {
System.out.printIn ("Error occured");
}}
```

Lab Report - 7

ResultSet is an object in JDBC that store data returned from a database query. It is mainly used with select queries. ResultSet works like a table with rows and columns. The next() method moves the cursor to the next row. It return true if data is avaible. The setString() method is used to read column by column from ResultSet. Resultset helps java programs fetch database records easily. Thus it is very important for retiring data in JDBC.

Java code: (resultSet usage)

```
import java.sql.*;
class resultset Example{
public state void main (string [] args) {
  try {
```

```
connection can = DriverManager.getConnection (
"Jdbc:mysql:// localhost:3306/test db", "root",
password",
statement st = cor, creat established ();
Result set as = st. execute query ("SELECT id,
        name from student");
while (rs.next) {
    int id = rs.getInt ("int");
    String name = rs.get string ("Name");
    System.out.println (id+" "+name);
  }
con.close ();
} catch (Exception e) {
System.out.println ("Error occured");
  }
 }
}
```

Lab Report - 8

Lab Tittle : Development of RESTful Web Service using Spring Boot.

objective: The objective of this lab is to understand how spring boot simplifies the development of RESTful web services and to implement a REST controller using @RestController, @GetMapping and @PostMapping annotation with JSON data handling.

Tool and Technologies used:

1. Java (JDK)
2. Spring boot Frame work
3. Spring web pependency
4. Embabed Apach Tomcat server

Theory: Spring boot greatly simpliefies the development of Restfull servies using.

@ post Mapping

i) Handless HTTP Post request

ii) used to sent data to the server.

Implementation:

Step 1: Creat Model class

```
Public class Student {
    private int id;
    private String name;
    public Student () { }
Public student ( int id, String name) {
        this. id = id;
        this. name = name;
    }
public int getId () {
        return id;
}
  public void setId (int id) {
        this. id = id;
}
  public string getName () {
      return name;
}
```

Get Response (list of student)

```
[
  {
    "id": 1,
    "name": "Rehim"
  }
]
```

Result: The RESTful web service was succesfully developed using spreing Boot. The application handled GET and POST request correctly and JSON data was atutomaticaly converted to and from Java objects.

Conclusion: Spring Boot simplifies REST ful web service development through auto-configuration. embedded servers and arrotation-based programming. Using @REST controller, @Get Mapping and @Post mapping. REST APIs can be created efficiently with aletomatic JSON handling.

Step-2 : Creat REST controller:

```
import org.springframework.web.bind.annotation.*;
import java.util.ArrayList;
import java.util.List;

@RESTcontroller
@RequestMapping("/students")
public class StudentController {
    private List<Student> students = new ArrayList<>();

    @Get mapping
    public List<Student> getAllStudents() {
        return students;
    }
    @Post mapping
    public student addstudent(@RequestBody Stude
        Student add(Student);
        returns student;
    }
}
```

Simple JSON Request and Response

Post Request (Add Student)

```
{ "id": 1,
  "name": "Rahim"
}
```

i) Auto configuration : Automatically configures web services and JSON converters with minimal set up.

ii) Embedded server : No need to deploy WARs to and external server. Just run your app like a Java application.

iii. Spring web starter: Includes all required dependencies for building REST APIs.

iv Reduced Boilerplate : Annotation like @ REST controller @ Get mapping @ postmapping etc. simplify request handling.

Description of Annotation:

@ REST controller:

i. Used to creat RESTful web services

ii. Combines @ controller and @ Response bonding

iii. Automatically returns data in JSON.

Java code:

```
abstract class vehicle {
abstract void star();
void fuelType() {
    system.out.println("uses fuel");
  }
}
```

Interface:

An interface is blueprint af a class that contains.

- Abstract methods
- constants (public static final)
- Defalit and static methods

Java code:

```
interface Electric {
  void change();
}
interface Autonomus {
  void autoDrive();
}
class tesla implements Electric, Autonomo
  Public void charg() {
    system.out.println("charging");
  }
```