Lab report No: 01

Key differences for multiple inheritance

Abstract classes:

- A class can only extend ONE abstract class
- Can have both abstract and concrete methods.
- can have instance variables with any access modifier
- Can have constructors
- methods can have any acess modifier (public, private, protect

Interfaces:

- A class can have multiple inheritance
- All methods are abstract by default
- Can only have constants
- Cannot have constructors
- All methods are implicitly public.

When to use Abstract classes

i) When we need to share code among related classes.

ii) we need non-public members

iii) we want to provide common functionality with some flexibility.

When to use interfaces

i) We need multiple inheritance

ii) Unrelated classes should implement the same methods.

iii) We want to specify a constract without any implementation.

Lab nepont No: 02

How encapsulation ensunes data secunity and Integnity.

Data secunity:

▷ Pnivate vaniables hide data fnom dinect extennal access.

▷ Only the class itself dinectly modify its data.

▷ Pnevents unauthonized on accidental changes from outside code.

Data Integnity:

▷ Validated setten methods ensune only valid data is stoned.

▷ Business nules ane enfonced befone data is changed.

▷ Invalid data is nejected, keeping the object in a valid state.

BankAccount Class Enample;

```
class BankAccount {
    pnivate String accountNumben;
    pnivate double balanée;
public void setAccountNumben (String accNO) {
    if(accNO!=null && !accNo.isEmpty()){
```

```
        accountNumber = accNo;
    }
}
public void setInitialBalance (double amount){
    if (amount >= 0) {
        balance = amount;
    }
}
public String getAccountNumber(){
    return accountNumber;
}
public double getBalance () {
    return balance;
}
}
```

Lab report No:03

```
import java.util.*;

    class RegistnanParking {

        String CanNo;
    RegistnanParking (String CanNo) { this.canNo = CanNo; }

     }

    class ParkingPool {

        Queue < RegistnaParking > queue = new LinkedList<>();

    synchronized void addRequest (Registnan Parking n){

            queue.add(n);

             notify();

        }

    class ParkingAgent extends Thread {

            ParkingPool pool;

            int agentId;
```

```
public void run(){
        try {
        RegistrarParking r= pool.getRequest();
    System.out.println("Agent"+ agentId + "parked car" +
                                    r.carNo + ".");

        } catch (Exception e) {}
    }
  }
public class mainclass {
        public static void main(String[] args){
        ParkingPool pool = new ParkingPool();

        }
    }
```

Lab report No: 04

How JDBC Manages Communication Between Java and Database

JDBC Connection:

JDBC (Java Database Connectivity) is an API that enables Java applications to interact with relational databases. It acts as a bridge between Java programs and database-specific drives, allowing execution of SQL queries and retrival of results in a platform-independent manner.

Steps to Execute a select query using JDBC:

i) import JDBC packages.

ii) Load and register the JDBC driver.

iii) Establish a database connection

iv) Create a statement.

v) Execute the SELECT query.

vi) Process the Resultset.

vii) close all resources.

Lab report NO: 05

Servlet Controller Role (MVC Pattern)

In a Java EE application following the MVC architecture:

▷ Model: Contains business logic and data.

▷ View: Jsp rendens the response to the client.

▷ Controller (servlet): Handles client requests, interacts wi
the model, and forwards data to the view.

Flow of Execution:

i) Client sends a request to the servlet.

ii) servlet processes the request and interact with the mo

iii) Servlet stores data in request scope.

iv) Request is forwarded to a Jsp.

v) Jsp rendens the response using the forwarded
data.

Lab report NO: 06

How prepared statement Improves performance and security over statement in JDBC.

Performance:

▷ Prepared statement is precompiled by the database.

▷ The same SQL query can be executed multiple times with different parameters without recompilation.

▷ This reduces parsing and execution time, especially for repeated operations.

Security:

▷ Prepared statement uses parameterized queries.

▷ User input is treated strictly as data, not an executable SQL.

▷ This effectively prevents SQL Injection attacks, which are possible with statement.

Example:

```
import java.sql.connection;
impont java.sql.DrivenManggen;
impont java.sql.PreparedStatement;

publie class InsentExample {
    public static void main(String[] args) thnows Exeeption{
        String url = "jdbc:mysql:// local host:3306/testdb";
        String usen = "noot";
        String pass = "password";

    Connection con = DrivenMaragen.getConnection(url, usen, pass);

    String sql = "INSERT INTO STUDENT (id, nane, age)
                  values (?,?,?)";

    PreparedStatement ps = con.preparedstatenent(sql);

    ps.setInt(1, 1);
    ps.setString(2, "Rahman");
    ps.setInt(3, 21);
    ps.executeUpdate();
     psrclose();
    con.close();
  }
}
```

Lab report No: 07

Resultset in JDBC

A resultset in JDBC is an object that stores the data retrieved from a database after executing a SQL SELECT query. It represents a table of data where the cursor initially points before the first row. Data is accessed row by row using cursor movement methods.

use of next(), getString(), getInt() methods with example

```
while (rs.next()) {
        int id = rs.getInt("id");
        String name = rs.getString("name");
        int age = rs.getInt("age");
        System.out.println(id + " " + name + " " + age);
}
```

Lab report No: 08

Spring Boot simplifies the development of RESTful services by praiding;

Auto-Configuration, which reduces manual setup and the configuration.

Embedded servers (such as Tomcat), eliminating the need for external deployment.

Annotation-based programming, making REST APIs easier to develop and maintain.

Automatic JSON conversion using the Jackson library.

Implementing a REST Controller in spring Boot:

```
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/student")
public class StudentController {
        @GetMapping("/{id}")
        public Student getStudent(@PathVariable int id) {
            return new Student(id, "Rahman", 21);
        }
```

```
@PostMapping("/add")
public Student addStudent(@RequestBody Student student){
    return student;
}
```

```
public class Student {
    private int id;
    private int age;
    private String name;

    public Student(int id, String name, int age) {
        this.id = id;
        this.age = age;
        this.name = name;
    }
}
```

Lab report NO:09

Important 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), textvariable
    = display, bd=10, insertwidth=4, width=14
    borderwidth=4, justify='right')

display_field.grid(row=0, column=0, columnpan=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.set("")
```

4. GUI Button Layout

```
Button(root, text='c', font=('arial', 14), command=clean,
width=5).grid(row=1, column=0, padx=5, pady=5)

Button(root, text='/', font=('arial', 14), command=lambda:
press('/'), width=5).grid(row=1, column=3, padx=5,
pady=5)

buttons = [
    ('7',2,0), ('8',2,1), ('9',2,2), ('*',2,3),
    ('4',3,0), ('5',3,1), ('6',3,2), ('-',3,3),
    ('1',4,0), ('2',4,1), ('3',4,2), ('+',4,3),
]
```

5. main Loop

```
root.mainloop()
```