

1. In Java, a protected member has special accessibility rules depending on inheritance and packages.
- child class in the same package:
- it can access protected members directly.

```
package pack1;  
  
class parent {  
    protected int n = 100;  
  
    protected void show() {  
        System.out.println("n = " + n);  
    }  
}  
  
class child extends parent {  
    void display() {  
        System.out.println("n = " + n); // accessible via inheritance  
        show(); // accessible  
    }  
}  
  
public static void main(String[] args) {  
    child obj = new child();  
    obj.display();  
}
```

2. Feature	Abstract class	Interface
multiple inheritance	Not supported	Supported (a class can implement multiple interfaces)
keyword	extends	implements
variables	instance variables allowed	only public static final constants.
constructor	allowed	not allowed
Access modifiers	public, private, protected	public by default.
purpose	share common base behavior	define capability or contract.

abstract class A {

 abstract void showA();

}

interface A {

 void showA();

}

abstract class B {

 abstract void showB();

}

interface B {

 void showB();

}

class C implements A, B {

 class C extends A, B {

}

}

Error

possible multiple inheritances

8. Encapsulation protects data by declaring variables as private and allowing access through validated public methods. It prevents unauthorized modification and ensures only valid data is stored. In a BankAccount class, setters validate account number and balance, rejecting null, empty, or negative values, thus maintaining data security and integrity.

```
class BankAccount {  
    private String accNo;  
    private double balance;  
  
    public void setAccountNumber (String accNo) {  
        if (accNo == null || accNo.trim().isEmpty()) {  
            System.out ("Invalid acc Number");  
        } else {  
            this.accountNumber = accNo;  
        }  
    }  
}
```

```
public void setInitialBalance (double amount) {  
    if (amount < 0) {  
        cout ("Balance cannot be neg");  
    } else {  
        this.balance = amount;  
    }  
}
```

```
public class main{
```

```
    public static void main (String args) {  
        BankAccount acc = new BankAccount();  
    }  
}
```

```
> } // end of main class
```

```
> } // end of BankAccount class
```

```
> } // end of BankAccount class
```

4.

(i) Find the kth smallest element in an ArrayList.

```
import java.util.*;  
public class kthsmallest {  
    public static void main (String [] args) {  
        ArrayList<Integer> list = new ArrayList<>(  
            Arrays.asList(10, 4, 7, 2, 15, 9));  
        int k = 3; // kth smallest element  
        Collections.sort(list);  
        System.out.println("sorted list: " + list);  
        System.out.println(k + "nd smallest element = " + list.get(k - 1));  
    }  
}
```

(iv) check if two Linked Lists are Equal

```
import java.util.*;  
public class LinkedListEqual {  
    public static void main (String [] args) {  
        LinkedList<Integer> list1 = new LinkedList<>(  
            Arrays.asList(1, 2, 3));
```

```

    LinkedList<Integer> list2 = new LinkedList<>(
        Arrays.asList(1, 2, 3));
    if (list1.equals(list2))
        cout ("lists are equal");
    else
        cout ("not equal");
}

```

(ii) TreeMap - Word frequency Counter

```

import java.util.*;
public class Wordfrequency {
    public void psvm() {
        String text = "java is easy and powerful";
        TreeMap<String, Integer> map = new TreeMap<>();
        String words[] = text.split(" ");
        for (String word : words) {
            map.put(word, map.getOrDefault(word, 0) + 1);
        }
        cout ("word frequencies:");
        cout (map);
    }
}

```

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 access modifier (public, private, protected)

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 construct without any implementation.

Lab report No: 02

How encapsulation ensures data security and Integrity .

Data security:

- ▷ Private variables hide data from direct external access.
- ▷ Only the class itself directly modify its data .
- ▷ Prevents unauthorized or accidental changes from outside code .

Data Integrity:

- ▷ Validated setter methods ensure only valid data is stored.
- ▷ Business rules are enforced before data is changed .
- ▷ Invalid data is rejected , keeping the object in a valid state .

BankAccount Class Example;

```
class BankAccount {  
    private String accountNumber;  
    private double balance;  
    public void setAccountNumber(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 RegistrationParking {  
    String carNo;  
    RegistrationParking (String carNo) { this.carNo = carNo; }  
}  
  
class ParkingPool {  
    Queue<RegistrationParking> queue = new LinkedList<>();  
    synchronized void addRequest (RegistrationParking n){  
        queue.add(n);  
        notify();  
    }  
  
    class ParkingAgent extends Thread {  
        parkingPool pool;  
        int agentId;
```

```
public void run() {
    try {
        RegistrationParking 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();
    }
}
```

Labs 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 drivers, allowing execution of SQL queries and retrieval 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 renders the response to the client.
- ▷ Controller (Servlet): Handles client requests, interacts with 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 model.
- iii) Servlet stores data in request scope.
- iv) Request is forwarded to a JSP.
- v) Jsp renders 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 as executable SQL.
- ▷ This effectively prevents SQL injection attacks, which are possible with statement.

Example:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class InsertExample {

    public static void main(String[] args) throws Exception {
        String url = "jdbc:mysql://localhost:3306/testdb";
        String user = "root";
        String pass = "password";

        Connection con = DriverManager.getConnection(url, user, pass);

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

        PreparedStatement ps = con.prepareStatement(sql);

        ps.setInt(1, 1);
        ps.setString(2, "Rahman");
        ps.setInt(3, 21);
        ps.executeUpdate();

        ps.close();
        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 providing:

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, "Prahmar", 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=15,
borderwidth=4, justify = 'right')
```

```
display-field.grid (row=0, column=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.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=2, 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),
]

6

5. Main Loop

root.mainloop()