ACADEMIC YEAR: 2024-25

Name & Register No of the Candidate: POTTRIVENDHAN C / 717824i240						
Course Code & Title: 23CSR306 JAVA PROGRAMMING						
Date of Issue: 4.09.2025			**Date of Submission:09.09.2025**			
Year/Dept./Sem/Section: II / AD / III / B						
Assignment: I						
Reference(s): BY GIVEN NOTES AND BOOK						
Marks Details						
Q. No						**Total (100)**
COs	CO1	CO2				
Marks Obtained						

Course In-charge

40. Design and implement a console-based OKR & Performance Review system to create

objectives, add key results, collect manager reviews, and generate final ratings using OOP in

Java.

Requirements:

1. Create at least 4 classes:

o Employee – empId, name, role, dept, reviewer.

o Objective – objId, title, description, weightage, status.

o KeyResult – krId, objectiveId, metric, target, progress.

o ReviewService – manages OKRs, updates progress, consolidates ratings.

2. Each class must include:

o ≥4 instance/static variables.

o A constructor to initialize values.

o ≥5 methods (getters/setters, addObjective(), addKeyResult(), updateProgress(), finalRating()).

3. Demonstrate OOPS Concepts:

o Inheritance → Manager extends Employee with approval privileges.

o Method Overloading → updateProgress() by value or by percent.

o Method Overriding → finalRating() rules differ for roles/levels.

o Polymorphism → store all people as Employee and dispatch role behavior.

o Encapsulation → protect rating edits and OKR weights via methods.

4. Write a Main class (OKRAppMain) to test:

o Create employees/OKRs, update KRs, submit/approve reviews.

o Print employee scorecards and department-wise rating distributions.

SOURCE CODE:

```
package okrsystem;

import java.util.*;
import java.util.stream.Collectors;

public class OKRAppMain {

    // ------ KeyResult ------
    static class KeyResult {
        private String krId;
        private String objectiveId;
        private String metric;
        private double target;
        private double progress;

        public KeyResult(String krId, String objectiveId, String metric, double target, double
progress) {
            this.krId = krId;
            this.objectiveId = objectiveId;
            this.metric = metric;
            this.target = Math.max(0, target);
            this.progress = clampProgress(progress);
        }

        public String getKrId() { return krId; }
        public String getObjectiveId() { return objectiveId; }
        public String getMetric() { return metric; }
        public double getTarget() { return target; }
        public double getProgress() { return progress; }
```

```
        public void setProgress(double progress) { this.progress =
    clampProgress(progress); }

        public double getPercentComplete() {
            if (target <= 0) return 0.0;
            return Math.min(100.0, (progress / target) * 100.0);
        }

        private double clampProgress(double p) {
            if (p < 0) return 0.0;
            if (p > target) return target;
            return p;
        }

        public String toString() {
            return String.format("KR[%s] metric=%s target=%.2f progress=%.2f (%.1f%%)",
                    krId, metric, target, progress, getPercentComplete());
        }
    }

    // ------ Objective ------
    static class Objective {
        private String objId;
        private String title;
        private String description;
        private double weightage;
        private String status;
        private boolean approved;
        private List<KeyResult> keyResults = new ArrayList<>();
```

```
    public Objective(String objId, String title, String description, double weightage, String
status) {
        this.objId = objId;
        this.title = title;
        this.description = description;
        setWeightage(weightage);
        this.status = status;
        this.approved = false;
    }

    public String getObjId() { return objId; }
    public String getTitle() { return title; }
    public double getWeightage() { return weightage; }
    public String getStatus() { return status; }
    public boolean isApproved() { return approved; }

    public void setWeightage(double weightage) {
        if (weightage < 0) this.weightage = 0;
        else if (weightage > 100) this.weightage = 100;
        else this.weightage = weightage;
    }

    public void addKeyResult(KeyResult kr) {
        if (kr != null && kr.getObjectiveId().equals(this.objId)) {
            keyResults.add(kr);
        }
    }

    public List<KeyResult> getKeyResults() { return
Collections.unmodifiableList(keyResults); }
```

```
        public double computeAverageKRPercent() {
            if (keyResults.isEmpty()) return 0.0;
            double sum = 0.0;
            for (KeyResult kr : keyResults) sum += kr.getPercentComplete();
            return sum / keyResults.size();
        }

        public void approveByManager(Manager m) { this.approved = true; }

        public String toString() {
            return String.format("Objective[%s] %s (weight=%.1f%%) status=%s approved=%s",
                    objId, title, weightage, status, approved);
        }
    }

    // ------ Employee ------
    static class Employee {
        protected String empId;
        protected String name;
        protected String role;
        protected String dept;
        protected Employee reviewer;
        private double rating = -1.0;
        protected List<Objective> objectives = new ArrayList<>();

        public Employee(String empId, String name, String role, String dept, Employee reviewer) {
            this.empId = empId;
            this.name = name;
            this.role = role;
```

```
        this.dept = dept;
        this.reviewer = reviewer;
    }

    public String getEmpId() { return empId; }
    public String getName() { return name; }
    public String getDept() { return dept; }

    public void addObjective(Objective o) { objectives.add(o); }
    public List<Objective> getObjectives() { return
Collections.unmodifiableList(objectives); }

    public double finalRating() {
        if (objectives.isEmpty()) return 0.0;
        double totalWeight = 0.0, weightedScoreSum = 0.0;
        for (Objective o : objectives) {
            double objWeight = o.getWeightage();
            double avgKRPercent = o.computeAverageKRPercent();
            weightedScoreSum += objWeight * avgKRPercent;
            totalWeight += objWeight;
        }
        if (totalWeight <= 0) return 0.0;
        double weightedAvgPercent = weightedScoreSum / totalWeight;
        double ratingValue = weightedAvgPercent / 20.0;
        ratingValue = Math.max(0.0, Math.min(5.0, ratingValue));
        setRating(ratingValue);
        return ratingValue;
    }

    protected void setRating(double rating) {
        if (rating < 0) rating = 0;
```

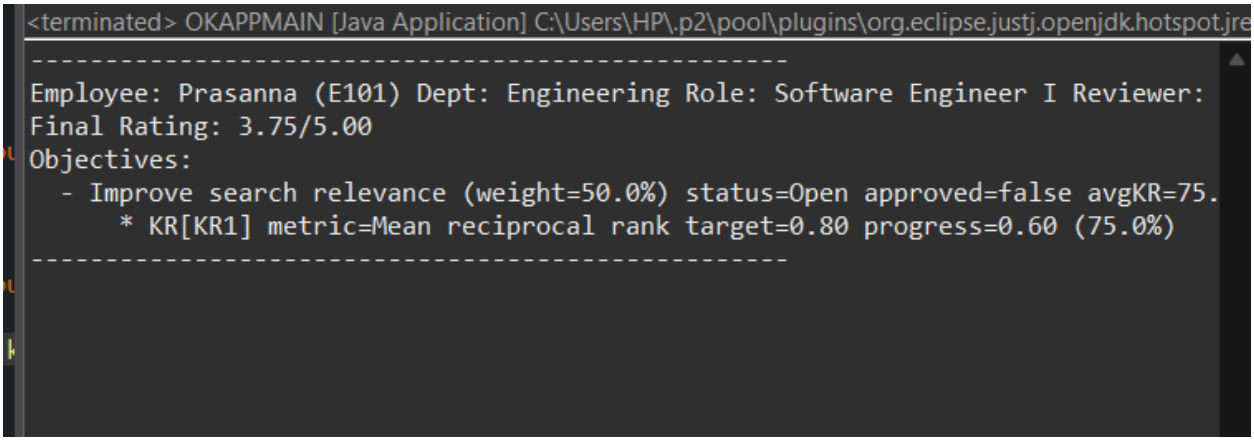
```
            if (rating > 5) rating = 5;
            this.rating = rating;
        }

        public double getRating() { return rating; }

        public void printScorecard() {
            System.out.println("------------------------------");
            System.out.printf("Employee: %s (%s) Dept: %s Role: %s Reviewer: %s\n",
                    name, empId, dept, role, reviewer != null ? reviewer.getName() : "None");
            System.out.printf("Final Rating: %.2f/5.00\n", getRating());
            System.out.println("Objectives:");
            for (Objective o : objectives) {
                System.out.printf("  - %s (weight=%.1f%%) status=%s approved=%s
avgKR=%.1f%%\n",
                        o.getTitle(), o.getWeightage(), o.getStatus(), o.isApproved(),
o.computeAverageKRPercent());
                for (KeyResult kr : o.getKeyResults()) {
                    System.out.printf("    * %s\n", kr);
                }
            }
            System.out.println("------------------------------");
        }
    }

    // ------ Manager ----------
    static class Manager extends Employee {
        private int level;
        private List<Employee> team = new ArrayList<>();
```

```
        public Manager(String empId, String name, String role, String dept, Employee reviewer, int level) {
            super(empId, name, role, dept, reviewer);
            this.level = Math.max(1, level);
        }

        public void addTeamMember(Employee e) { if (e != null) team.add(e); }
        public void approveObjective(Objective o) { if (o != null) o.approveByManager(this); }

        @Override
        public double finalRating() {
            double base = super.finalRating();
            double bonus = (level - 1) * 0.1;
            double adjusted = Math.min(5.0, base + bonus);
            setRating(adjusted);
            return adjusted;
        }
    }

    // ---------- ReviewService ----------
    static class ReviewService {
        private Map<String, Employee> employees = new HashMap<>();
        private Map<String, Objective> objectives = new HashMap<>();
        private Map<String, KeyResult> keyResults = new HashMap<>();

        public void registerEmployee(Employee e) { employees.put(e.getEmpId(), e); }
        public void addObjectiveToEmployee(String empId, Objective o) {
            Employee e = employees.get(empId);
            e.addObjective(o);
            objectives.put(o.getObjId(), o);
```

```
        }
        public void addKeyResult(KeyResult kr) {
            Objective o = objectives.get(kr.getObjectiveId());
            o.addKeyResult(kr);
            keyResults.put(kr.getKrId(), kr);
        }

        public void updateProgress(String krId, double newProgressValue) {
            KeyResult kr = keyResults.get(krId);
            kr.setProgress(newProgressValue);
        }
        public void updateProgress(String krId, double percent, boolean isPercent) {
            KeyResult kr = keyResults.get(krId);
            double newValue = (percent / 100.0) * kr.getTarget();
            kr.setProgress(newValue);
        }

        public void consolidateAllRatings() {
            for (Employee e : employees.values()) e.finalRating();
        }

        public void printEmployeeScorecard(String empId) {
            employees.get(empId).printScorecard();
        }
    }

    // ------ MAIN ------
    public static void main(String[] args) {
        ReviewService svc = new ReviewService();
        Manager gm = new Manager("M001", "Alice Johnson", "Engineering Manager",
"Engineering", null, 2);
```

```
        svc.registerEmployee(gm);

        Employee e1 = new Employee("E101", "Prasanna", "Software Engineer I",
"Engineering", gm);
        svc.registerEmployee(e1);

        Objective o1 = new Objective("O1-E101", "Improve search relevance", "Improve
results ranking quality", 50, "Open");
        svc.addObjectiveToEmployee(e1.getEmpId(),  o1);

        svc.addKeyResult(new KeyResult("KR1", o1.getObjId(), "Mean reciprocal rank",
0.8, 0.32));

        svc.updateProgress("KR1", 0.6);

        svc.consolidateAllRatings();
        svc.printEmployeeScorecard(e1.getEmpId());
    }
}
```

OUTPUT:



GITHUB LINK:

https://github.com/Pottrivendhan/Employee