**C-DAC Mumbai**

**OOPJ Lab**

**Assignment 5**

**Problem 1: Student Names Management System**

**Use Case:** A school administrator needs to maintain a list of student names for a class roster.

->import java.util.ArrayList;

public class StudentNamesManagement {

public static void main(String[] args) {

ArrayList<String> students = new ArrayList<>();

students.add("Amit");

students.add("Priya");

students.add("Rohan");

students.remove("Priya");

System.out.print("Students: ");

for (int i = 0; i < students.size(); i++) {

System.out.print(students.get(i));

if (i < students.size() - 1) System.out.print(", ");

}

}

}

**Problem 2: Lab Access Queue System**

**Use Case:** A computer lab needs to manage students waiting for access using a first-come-first-served system.

**->**

import java.util.LinkedList;

import java.util.Queue;

public class LabAccessQueueSystem {

public static void main(String[] args) {

// Create a Queue to store students

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

// Enqueue students

queue.offer("Amit");

queue.offer("Priya");

queue.offer("Rohan");

// Dequeue one student (first-come-first-served)

queue.poll();

// Display remaining queue

System.out.print("Queue: ");

int count = 0;

for (String student : queue) {

System.out.print(student);

count++;

if (count < queue.size()) System.out.print(", ");

}

}

}

Problem 3: Daily Task Tracker

Use Case: A student wants to track their daily tasks and mark completed ones.

->import java.util.ArrayList;

public class DailyTaskTracker {

public static void main(String[] args) {

ArrayList<String> tasks = new ArrayList<>();

// Adding tasks

tasks.add("Study Java");

tasks.add("Complete Assignment");

tasks.add("Exercise");

// Marking one task as completed (remove)

tasks.remove("Exercise");

// Display remaining tasks

System.out.print("Remaining tasks: ");

for (int i = 0; i < tasks.size(); i++) {

System.out.print(tasks.get(i));

if (i < tasks.size() - 1) System.out.print(", ");

}

}

}

Problem 4: Grocery Shopping List

Use Case: A person maintains a grocery list and removes items as they purchase them.

->import java.util.ArrayList;

public class GroceryList {

public static void main(String[] args) {

ArrayList<String> groceries = new ArrayList<>();

groceries.add("Milk");

groceries.add("Eggs");

groceries.add("Bread");

groceries.remove("Milk"); // Purchased

System.out.print("Items to buy: ");

for (int i = 0; i < groceries.size(); i++) {

System.out.print(groceries.get(i));

if (i < groceries.size() - 1) System.out.print(", ");

}

}

}

Problem 5: Recent Search History

Use Case: A search application maintains the last 5 searches, removing the oldest when the limit is exceeded.

->import java.util.LinkedList;

public class RecentSearchHistory {

public static void main(String[] args) {

LinkedList<String> searches = new LinkedList<>();

String[] input = {"Java", "Python", "C++", "DSA", "OOP", "Spring"};

for (String search : input) {

if (searches.size() == 5) { // limit exceeded → remove oldest

searches.removeFirst();

}

searches.addLast(search);

}

System.out.print("Recent searches: ");

for (int i = 0; i < searches.size(); i++) {

System.out.print(searches.get(i));

if (i < searches.size() - 1) System.out.print(", ");

}

}

}

Problem 6: Unique Roll Number Validator

Use Case: A registration system must ensure no duplicate roll numbers are assigned.

->import java.util.LinkedHashSet;

public class UniqueRollNumbers {

public static void main(String[] args) {

LinkedHashSet<Integer> rollNumbers = new LinkedHashSet<>();

int[] input = {101, 102, 101, 103};

for (int num : input) rollNumbers.add(num);

System.out.print("Unique Roll Numbers: ");

int count = 0;

for (int num : rollNumbers) {

System.out.print(num);

count++;

if (count < rollNumbers.size()) System.out.print(", ");

}

}

}

Problem 7: Alphabetical Student Directory

Use Case: A school wants to maintain student names in alphabetical order.

->import java.util.TreeSet;

public class AlphabeticalStudentDirectory {

public static void main(String[] args) {

TreeSet<String> students = new TreeSet<>();

students.add("Rohan");

students.add("Amit");

students.add("Priya");

System.out.print("Students: ");

int count = 0;

for (String name : students) {

System.out.print(name);

count++;

if (count < students.size()) System.out.print(", ");

}

}

}

Problem 8: Course Registration System

Use Case: A student registers for courses, ensuring no duplicate course codes.

->import java.util.LinkedHashSet;

public class CourseRegistration {

public static void main(String[] args) {

LinkedHashSet<String> courses = new LinkedHashSet<>();

String[] input = {"CS101", "MA101", "CS101"};

for (String code : input) courses.add(code);

System.out.print("Registered Courses: ");

int count = 0;

for (String course : courses) {

System.out.print(course);

count++;

if (count < courses.size()) System.out.print(", ");

}

}

}

**Problem 9: Event Attendance Counter**

Use Case: Count unique attendees at an event, handling duplicate check-ins.

->import java.util.HashSet;

public class UniqueAttendees {

public static void main(String[] args) {

HashSet<String> attendees = new HashSet<>();

String[] input = {"Amit", "Rohan", "Amit", "Priya"};

for (String name : input) attendees.add(name);

System.out.println("Total unique attendees: " + attendees.size());

}

}

Problem 10: Electronic Voting System

Use Case: Track unique voters in an election system to prevent duplicate voting.

->import java.util.HashSet;

public class ElectronicVotingSystem {

public static void main(String[] args) {

HashSet<Integer> voters = new HashSet<>();

int[] input = {201, 202, 203, 202};

for (int id : input) voters.add(id);

System.out.println("Total voters: " + voters.size());

}

}

Problem 11: Student Grade Management

Use Case: A teacher needs to map student names to their exam marks.

->import java.util.HashMap;

public class StudentGradeManagement {

public static void main(String[] args) {

HashMap<String, Integer> grades = new HashMap<>();

grades.put("Amit", 85);

grades.put("Priya", 92);

grades.put("Rohan", 78);

System.out.print("Grade Report: ");

int count = 0;

for (String name : grades.keySet()) {

System.out.print(name + ":" + grades.get(name));

count++;

if (count < grades.size()) System.out.print(", ");

}

}

}

**Problem 12: Attendance Tracking System**

**Use Case:** Track student attendance percentages in alphabetical order.

**->import java.util.Map;**

**import java.util.TreeMap;**

**public class AttendanceTrackingSystem {**

**public static void main(String[] args) {**

**// TreeMap automatically sorts keys in alphabetical order**

**TreeMap<String, Integer> attendance = new TreeMap<>();**

**// Adding sample data**

**attendance.put("Amit", 90);**

**attendance.put("Rohan", 85);**

**attendance.put("Priya", 95);**

**System.out.print("Attendance Report: ");**

**int count = 0;**

**for (Map.Entry<String, Integer> entry : attendance.entrySet()) {**

**System.out.print(entry.getKey() + ":" + entry.getValue());**

**count++;**

**if (count < attendance.size()) System.out.print(", ");**

**}**

**}**

**}**

**Problem 13: Student Registration Order Tracker**

**Use Case:** Maintain the order in which students registered for a course.

**->import java.util.LinkedHashMap;**

**import java.util.Map;**

**public class StudentRegistrationOrder {**

**public static void main(String[] args) {**

**// LinkedHashMap preserves insertion order**

**LinkedHashMap<String, Integer> registrations = new LinkedHashMap<>();**

**registrations.put("Amit", 101);**

**registrations.put("Rohan", 102);**

**registrations.put("Priya", 103);**

**System.out.print("Registration Order: ");**

**int count = 0;**

**for (Map.Entry<String, Integer> entry : registrations.entrySet()) {**

**System.out.print(entry.getKey() + ":" + entry.getValue());**

**count++;**

**if (count < registrations.size()) System.out.print(", ");**

**}**

**}**

**}**

**Problem 14: Grade Update System**

**Use Case:** Update a student's marks in the grading system.

->import java.util.HashMap;

public class GradeUpdateSystem {

public static void main(String[] args) {

HashMap<String, Integer> grades = new HashMap<>();

// Initial grade

grades.put("Rohan", 78);

// Update grade

grades.put("Rohan", 88); // put() replaces existing value for same key

System.out.println("Updated Grade: Rohan:" + grades.get("Rohan"));

}

}

**Problem 15: Library Book Inventory**

**Use Case: Track available copies of books in a library system.**

**->** **import java.util.HashMap;**

**import java.util.Map;**

**public class LibraryBookInventory {**

**public static void main(String[] args) {**

**HashMap<String, Integer> inventory = new HashMap<>();**

**// Initial inventory**

**inventory.put("Java", 3);**

**inventory.put("Python", 5);**

**// Borrow 1 copy of "Java"**

**String borrowedBook = "Java";**

**if (inventory.containsKey(borrowedBook)) {**

**inventory.put(borrowedBook, inventory.get(borrowedBook) - 1);**

**}**

**// Display current inventory**

**System.out.print("Current inventory: ");**

**int count = 0;**

**for (Map.Entry<String, Integer> entry : inventory.entrySet()) {**

**System.out.print(entry.getKey() + ":" + entry.getValue());**

**count++;**

**if (count < inventory.size()) System.out.print(", ");**

**}**

**}**

**}**

**Problem 16: Grade-Based Student Filter**

**Use Case: Remove students with failing grades from the honor roll.**

**->** **import java.util.HashMap;**

**import java.util.Iterator;**

**import java.util.Map;**

**public class GradeBasedStudentFilter {**

**public static void main(String[] args) {**

**HashMap<String, Integer> grades = new HashMap<>();**

**// Adding students and grades**

**grades.put("Amit", 85);**

**grades.put("Priya", 52);**

**grades.put("Rohan", 78);**

**// Filter out students below 60**

**Iterator<Map.Entry<String, Integer>> iterator = grades.entrySet().iterator();**

**while (iterator.hasNext()) {**

**if (iterator.next().getValue() < 60) {**

**iterator.remove();**

**}**

**}**

**// Display honor roll**

**System.out.print("Honor Roll: ");**

**int count = 0;**

**for (Map.Entry<String, Integer> entry : grades.entrySet()) {**

**System.out.print(entry.getKey() + ":" + entry.getValue());**

**count++;**

**if (count < grades.size()) System.out.print(", ");**

**}**

**}**

**}**

**Problem 17: Grade Distribution Counter**

**Use Case: Analyze the distribution of grades in a class.**

-> import java.util.HashMap;

import java.util.Map;

public class GradeDistributionCounter {

public static void main(String[] args) {

String[] grades = {"A", "B", "A", "C", "B", "A"};

HashMap<String, Integer> distribution = new HashMap<>();

// Count frequency of each grade

for (String grade : grades) {

distribution.put(grade, distribution.getOrDefault(grade, 0) + 1);

}

// Display grade distribution

System.out.print("Grade Distribution: ");

int count = 0;

for (Map.Entry<String, Integer> entry : distribution.entrySet()) {

System.out.print(entry.getKey() + "=" + entry.getValue());

count++;

if (count < distribution.size()) System.out.print(", ");

}

}

}

**Problem 18: Batch Merger System**

**Use Case:** Merge student lists from morning and evening batches, removing duplicates.

**->** **import java.util.HashSet;**

**public class BatchMergerSystem {**

**public static void main(String[] args) {**

**String[] morningBatch = {"Amit", "Priya"};**

**String[] eveningBatch = {"Rohan", "Priya"};**

**HashSet<String> combined = new HashSet<>();**

**// Add all students from both batches**

**for (String student : morningBatch) combined.add(student);**

**for (String student : eveningBatch) combined.add(student);**

**// Display combined batch (duplicates automatically removed)**

**System.out.print("Combined batches: ");**

**int count = 0;**

**for (String student : combined) {**

**System.out.print(student);**

**count++;**

**if (count < combined.size()) System.out.print(", ");**

**}**

**}**

**}**

**Problem 19: Grade Report Generator**

**Use Case:** Display all student grades using proper iteration techniques.

**->** **import java.util.HashMap;**

**import java.util.Iterator;**

**import java.util.Map;**

**import java.util.Set;**

**public class GradeReportGenerator {**

**public static void main(String[] args) {**

**HashMap<String, Integer> grades = new HashMap<>();**

**grades.put("Amit", 85);**

**grades.put("Priya", 92);**

**System.out.print("Grade Report: ");**

**// Use iterator to traverse entrySet**

**Set<Map.Entry<String, Integer>> entries = grades.entrySet();**

**Iterator<Map.Entry<String, Integer>> iterator = entries.iterator();**

**while (iterator.hasNext()) {**

**Map.Entry<String, Integer> entry = iterator.next();**

**System.out.print(entry.getKey() + ":" + entry.getValue());**

**if (iterator.hasNext()) System.out.print(", ");**

**}**

**}**

**}**

**Problem 20: Even Roll Number Filter**

**Use Case:** Filter and display only students with even roll numbers.

**->** **import java.util.ArrayList;**

**import java.util.Iterator;**

**public class EvenRollNumberFilter {**

**public static void main(String[] args) {**

**ArrayList<Integer> rollNumbers = new ArrayList<>();**

**rollNumbers.add(101);**

**rollNumbers.add(102);**

**rollNumbers.add(103);**

**rollNumbers.add(104);**

**// Remove odd roll numbers using iterator**

**Iterator<Integer> iterator = rollNumbers.iterator();**

**while (iterator.hasNext()) {**

**int roll = iterator.next();**

**if (roll % 2 != 0) iterator.remove();**

**}**

**System.out.print("Even Roll Numbers: ");**

**for (int i = 0; i < rollNumbers.size(); i++) {**

**System.out.print(rollNumbers.get(i));**

**if (i < rollNumbers.size() - 1) System.out.print(", ");**

**}**

**}**

**}**

**Problem 21: Text Editor Undo System**

**Use Case:** Implement undo functionality for a text editor.

-> import java.util.Stack;

public class TextEditorUndoSystem {

public static void main(String[] args) {

Stack<String> actions = new Stack<>();

// Add actions

actions.push("Type A");

actions.push("Type B");

actions.push("Delete");

// Undo last action

if (!actions.isEmpty()) {

System.out.println("Undoing: " + actions.pop());

}

// Display current actions

System.out.print("Current actions: ");

for (int i = 0; i < actions.size(); i++) {

System.out.print(actions.get(i));

if (i < actions.size() - 1) System.out.print(", ");

}

}

}

**Problem 22: Ticket Booking Queue**

**Use Case:** Manage customer service in a ticket booking system.

-> import java.util.LinkedList;

import java.util.Queue;

public class TicketBookingQueue {

public static void main(String[] args) {

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

// Add customers

queue.add("Amit");

queue.add("Priya");

queue.add("Rohan");

// Serve 1 customer

String serving = queue.poll(); // removes head of queue

System.out.print("Serving: " + serving + ", ");

// Display remaining queue

System.out.print("Queue: ");

int count = 0;

for (String customer : queue) {

System.out.print(customer);

count++;

if (count < queue.size()) System.out.print(", ");

}

}

}

**Problem 23: Browser History Management**

**Use Case:** Maintain browser history with back functionality.

**->** **import java.util.Stack;**

public class BrowserHistoryManagement {

public static void main(String[] args) {

Stack<String> history = new Stack<>();

// Visiting pages

history.push("Google");

history.push("YouTube");

history.push("GFG");

// Go back by 1 page

if (!history.isEmpty()) {

System.out.println("Going back from: " + history.pop());

}

// Display current history

System.out.print("Current history: ");

for (int i = 0; i < history.size(); i++) {

System.out.print(history.get(i));

if (i < history.size() - 1) System.out.print(", ");

}

}

**}**

**Problem 24: Print Job Queue Manager**

**Use Case:** Manage print jobs in a shared printer system.

-> import java.util.LinkedList;

import java.util.Queue;

public class PrintJobQueueManager {

public static void main(String[] args) {

Queue<String> printJobs = new LinkedList<>();

// Adding jobs

printJobs.add("Doc1");

printJobs.add("Doc2");

printJobs.add("Doc3");

// Process one job

String printing = printJobs.poll();

System.out.print("Printing " + printing + ", Queue: ");

int count = 0;

for (String job : printJobs) {

System.out.print(job);

count++;

if (count < printJobs.size()) System.out.print(", ");

}

}

}

**Problem 25: Command History Tracker**

**Use Case:** Store recent commands in a terminal with limited history.

-> import java.util.LinkedList;

public class CommandHistoryTracker {

public static void main(String[] args) {

int limit = 3;

LinkedList<String> history = new LinkedList<>();

// Add commands

addCommand(history, "ls", limit);

addCommand(history, "pwd", limit);

addCommand(history, "cd ..", limit);

addCommand(history, "mkdir", limit); // Exceeds limit -> removes "ls"

System.out.print("Recent Commands: ");

for (int i = 0; i < history.size(); i++) {

System.out.print(history.get(i));

if (i < history.size() - 1) System.out.print(", ");

}

}

static void addCommand(LinkedList<String> history, String command, int limit) {

if (history.size() == limit) {

history.removeFirst(); // remove oldest

}

history.addLast(command);

}

}

**Problem 26: Employee Management System Use Case: Manage** employee information including name and salary.

-> import java.util.ArrayList;

class Employee {

String name;

double salary;

Employee(String name, double salary) {

this.name = name;

this.salary = salary;

}

@Override

public String toString() {

return name + ":" + salary;

}

}

public class EmployeeManagementSystem {

public static void main(String[] args) {

ArrayList<Employee> employees = new ArrayList<>();

employees.add(new Employee("Amit", 50000));

employees.add(new Employee("Priya", 60000));

System.out.print("Employee List: ");

for (int i = 0; i < employees.size(); i++) {

System.out.print(employees.get(i));

if (i < employees.size() - 1) System.out.print(", ");

}

}

}

**Problem 27: Employee Salary Sorting Use Case:** Sort employees by salary for payroll processing.

-> import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

public class EmployeeSalarySorting {

public static void main(String[] args) {

ArrayList<Employee> employees = new ArrayList<>();

employees.add(new Employee("Amit", 50000));

employees.add(new Employee("Priya", 60000));

employees.add(new Employee("Rohan", 45000));

Collections.sort(employees, Comparator.comparingDouble(emp -> emp.salary));

System.out.print("Sorted by salary: ");

for (int i = 0; i < employees.size(); i++) {

System.out.print(employees.get(i));

if (i < employees.size() - 1) System.out.print(", ");

}

}

}

**Problem 28: Department Employee Mapping Use Case:** Organize employees by department for HR management.

-> import java.util.\*;

public class DepartmentEmployeeMapping {

public static void main(String[] args) {

Map<String, List<String>> departmentMap = new HashMap<>();

departmentMap.put("IT", new ArrayList<>(List.of("Amit", "Rohan")));

departmentMap.put("HR", new ArrayList<>(List.of("Priya")));

System.out.print("Department Structure: ");

int deptCount = 0;

for (Map.Entry<String, List<String>> entry : departmentMap.entrySet()) {

System.out.print(entry.getKey() + ": " + String.join(", ", entry.getValue()));

deptCount++;

if (deptCount < departmentMap.size()) System.out.print("; ");

}

}

}

**Problem 29: Student Record System Use Case:** Maintain student records with name and grade information.

-> import java.util.ArrayList;

class Student {

String name;

String grade;

Student(String name, String grade) {

this.name = name;

this.grade = grade;

}

@Override

public String toString() {

return name + ":" + grade;

}

}

public class StudentRecordSystem {

public static void main(String[] args) {

ArrayList<Student> students = new ArrayList<>();

students.add(new Student("Amit", "A"));

students.add(new Student("Priya", "B"));

System.out.print("Student Records: ");

for (int i = 0; i < students.size(); i++) {

System.out.print(students.get(i));

if (i < students.size() - 1) System.out.print(", ");

}

}

}

**Problem 30: Grade-Based Student Filter Use Case:** Filter students based on minimum grade requirements

-> import java.util.ArrayList;

import java.util.Iterator;

public class GradeBasedStudentFilter1 {

public static void main(String[] args) {

ArrayList<Student> students = new ArrayList<>();

students.add(new Student("Amit", "A"));

students.add(new Student("Priya", "C"));

students.add(new Student("Rohan", "B"));

// Filter students with grade < B

Iterator<Student> iterator = students.iterator();

while (iterator.hasNext()) {

Student s = iterator.next();

if (s.grade.compareTo("B") > 0) { // "C" > "B"

iterator.remove();

}

}

System.out.print("Qualified Students: ");

for (int i = 0; i < students.size(); i++) {

System.out.print(students.get(i));

if (i < students.size() - 1) System.out.print(", ");

}

}

}

**University Student Management System Placement Pakka Problem** **Statement**: Create a University Student Management System using Java Collections to manage students across departments.

● Roll Number (int)

● Name (String)

● Department (String)

● CGPA (double)

-> import java.util.\*;

// Student class with Comparable for Merit List

class Student implements Comparable<Student> {

int rollNo;

String name;

String department;

double cgpa;

Student(int rollNo, String name, String department, double cgpa) {

this.rollNo = rollNo;

this.name = name;

this.department = department;

this.cgpa = cgpa;

}

@Override

public int compareTo(Student other) {

// Sort by CGPA (descending)

return Double.compare(other.cgpa, this.cgpa);

}

@Override

public String toString() {

return name + "(" + cgpa + ")";

}

}

public class UniversityStudentManagementSystem {

public static void main(String[] args) {

Student s1 = new Student(101, "Amit", "CS", 8.5);

Student s2 = new Student(102, "Priya", "Math", 9.2);

Student s3 = new Student(103, "Rohan", "CS", 7.8);

Student s4 = new Student(104, "Sneha", "Physics", 4.5);

// REGISTRATION LIST (ArrayList)

List<Student> registrationList = new ArrayList<>();

Collections.addAll(registrationList, s1, s2, s3, s4);

System.out.println("Registration Order: " + getNames(registrationList));

// MERIT LIST (Comparable - Sort by CGPA descending)

List<Student> meritList = new ArrayList<>(registrationList);

Collections.sort(meritList);

System.out.println("Merit List: " + meritList);

// ALPHABETICAL LIST (Comparator)

List<Student> alphaList = new ArrayList<>(registrationList);

alphaList.sort(Comparator.comparing(st -> st.name));

System.out.println("Alphabetical: " + getNames(alphaList));

// DEPARTMENT GROUPING (HashMap)

Map<String, List<Student>> deptGroups = new HashMap<>();

for (Student s : registrationList) {

deptGroups.computeIfAbsent(s.department, k -> new ArrayList<>()).add(s);

}

System.out.println("Department Groups:");

for (Map.Entry<String, List<Student>> entry : deptGroups.entrySet()) {

System.out.println(entry.getKey() + ": " + getNames(entry.getValue()));

}

// UNIQUE NAMES (HashSet)

Set<String> uniqueNames = new HashSet<>();

for (Student s : registrationList) uniqueNames.add(s.name);

System.out.println("Unique Names: " + uniqueNames);

// ROLL NUMBER SORTING (TreeSet)

Set<Integer> rollNoSet = new TreeSet<>();

for (Student s : registrationList) rollNoSet.add(s.rollNo);

System.out.println("Sorted Roll Numbers: " + rollNoSet);

// PERFORMANCE FILTER (Iterator)

List<Student> filteredList = new ArrayList<>(registrationList);

Iterator<Student> iterator = filteredList.iterator();

while (iterator.hasNext()) {

if (iterator.next().cgpa < 5.0) {

iterator.remove();

}

}

System.out.println("After Filter (CGPA >= 5.0): " + getNames(filteredList));

// RECENT REGISTRATIONS (Stack - LIFO)

Stack<Student> recentStack = new Stack<>();

recentStack.push(s1);

recentStack.push(s2);

recentStack.push(s3);

System.out.println("Recent Registrations (Top): " + recentStack.peek());

// SCHOLARSHIP QUEUE (FIFO)

Queue<Student> scholarshipQueue = new LinkedList<>();

scholarshipQueue.add(s1);

scholarshipQueue.add(s2);

System.out.println("Scholarship Processing (FIFO): " + scholarshipQueue.poll().name);

// HOSTEL APPLICATIONS (LinkedList)

LinkedList<Integer> hostelQueue = new LinkedList<>();

hostelQueue.addLast(105); // Regular applicant

System.out.println("Hostel Queue After Regular: " + hostelQueue);

hostelQueue.addFirst(101); // Priority applicant

System.out.println("Hostel Queue After Priority: " + hostelQueue);

hostelQueue.removeFirst(); // Remove from front

System.out.println("Hostel Queue After Allocation: " + hostelQueue);

}

// Utility method to get just names

private static String getNames(List<Student> list) {

List<String> names = new ArrayList<>();

for (Student s : list) names.add(s.name);

return names.toString();

}

}