C-DAC Mumbai
OOPJ Lab Assignment

Problem 1: Student Names Management System
Use Case: A school administrator needs to maintain a list of student names for a class roster.
Requirements:
- Add student names to the roster
- Display all students
- Remove a student from the roster

Sample Input:
Add students: "Amit", "Priya", "Rohan"
Remove student: "Priya"
Expected Output:
Students: Amit, Rohan

```java
import java.util.*;

public class StudentRoster {
    public static void main(String[] args) {
        List<String> students = new ArrayList<>();
        students.add("Amit");
        students.add("Priya");
        students.add("Rohan");
        students.remove("Priya");
        System.out.println("Students: " + String.join(", ", students));
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 5> javac StudentRoster.java
PS C:\Users\baenu\Test\OOPJ Assignment 5> java StudentRoster
Students: Amit, Rohan
```

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.
Requirements:
- Students join the queue for lab access
- Process students in FIFO order
- Display remaining queue

Sample Input:
Enqueue: "Amit", "Priya", "Rohan"
Dequeue: 1 student
Expected Output:
Queue: Priya, Rohan

```java
import java.util.*;

public class LabQueue {
    public static void main(String[] args) {
        Queue<String> queue = new LinkedList<>();
        queue.add("Amit");
        queue.add("Priya");
        queue.add("Rohan");
        queue.poll();
        System.out.println("Queue: " + String.join(", ", queue));
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 5> javac LabQueue.java
PS C:\Users\baenu\Test\OOPJ Assignment 5> java LabQueue
Queue: Priya, Rohan
```

Problem 3: Daily Task Tracker
Use Case: A student wants to track their daily tasks and mark completed ones.
Requirements:
- Add tasks to the list
- Mark tasks as completed (remove them)
- Display remaining tasks

Sample Input:
Add tasks: "Study Java", "Complete Assignment", "Exercise"
Complete task: "Exercise"
Expected Output:
Remaining tasks: Study Java, Complete Assignment

```java
import java.util.*;

public class TaskTracker {
    public static void main(String[] args) {
        List<String> tasks = new ArrayList<>();
        tasks.add("Study Java");
        tasks.add("Complete Assignment");
        tasks.add("Exercise");
        tasks.remove("Exercise");
        System.out.println("Remaining tasks: " + String.join(", ", tasks));
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 5> javac TaskTracker.java
PS C:\Users\baenu\Test\OOPJ Assignment 5> java TaskTracker
Remaining tasks: Study Java, Complete Assignment
```

Problem 4: Grocery Shopping List
Use Case: A person maintains a grocery list and removes items as they purchase them.
Requirements:
● Add items to grocery list
● Remove purchased items
● Display remaining items

Sample Input:
Add items: "Milk", "Eggs", "Bread"
Purchase: "Milk"
Expected Output:
Items to buy: Eggs, Bread

```java
import java.util.*;

public class GroceryList {
    public static void main(String[] args) {
        List<String> items = new ArrayList<>();
        items.add("Milk");
        items.add("Eggs");
        items.add("Bread");
        items.remove("Milk");
        System.out.println("Items to buy: " + String.join(", ", items));
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 5> javac GroceryList.java
PS C:\Users\baenu\Test\OOPJ Assignment 5> java GroceryList
Items to buy: Eggs, Bread
```

Problem 5: Recent Search History
Use Case: A search application maintains the last 5 searches, removing the oldest when the limit is exceeded.
Requirements:
● Store recent searches (maximum 5)
● Remove oldest search when limit exceeded
● Maintain insertion order

Sample Input:
Searches: "Java", "Python", "C++", "DSA", "OOP", "Spring"
Expected Output:
Recent searches: Python, C++, DSA, OOP, Spring

```java
import java.util.*;

public class SearchHistory {
    public static void main(String[] args) {
        Queue<String> searches = new LinkedList<>();
        String[] inputs = {"Java", "Python", "C++", "DSA", "OOP", "Spring"};
        for (String s : inputs) {
            if (searches.size() == 5) searches.poll();
            searches.add(s);
        }
        System.out.println("Recent searches: " + String.join(", ", searches));
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 5> javac SearchHistory.java
PS C:\Users\baenu\Test\OOPJ Assignment 5> java SearchHistory
Recent searches: Python, C++, DSA, OOP, Spring
```

Problem 6: Unique Roll Number Validator
Use Case: A registration system must ensure no duplicate roll numbers are assigned.
Requirements:
- Accept roll numbers for registration
- Automatically remove duplicates
- Display unique roll numbers

Sample Input:
Roll numbers: 101, 102, 101, 103
Expected Output:
Unique Roll Numbers: 101, 102, 103

```java
import java.util.*;

public class RollValidator {
    public static void main(String[] args) {
        Set<Integer> rolls = new LinkedHashSet<>();
        int[] nums = {101, 102, 101, 103};
        for (int n : nums) rolls.add(n);
        System.out.println("Unique Roll Numbers: " + rolls);
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 5> javac RollValidator.java
PS C:\Users\baenu\Test\OOPJ Assignment 5> java RollValidator
Unique Roll Numbers: [101, 102, 103]
```

Problem 7: Alphabetical Student Directory
Use Case: A school wants to maintain student names in alphabetical order.
Requirements:
- Add student names
- Automatically maintain alphabetical sorting
- Display sorted name

Sample Input:
Students: "Rohan", "Amit", "Priya"
Expected Output:
Students: Amit, Priya, Rohan

```java
import java.util.*;

public class StudentDir {
    public static void main(String[] args) {
        Set<String> students = new TreeSet<>();
        students.add("Rohan");
        students.add("Amit");
        students.add("Priya");
        System.out.println("Students: " + String.join(", ", students));
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 5> javac StudentDir.java
PS C:\Users\baenu\Test\OOPJ Assignment 5> java StudentDir
Students: Amit, Priya, Rohan
```

Problem 8: Course Registration System
Use Case: A student registers for courses, ensuring no duplicate course codes.
Requirements:
- Register for courses using course codes
- Prevent duplicate registrations
- Display registered courses

Sample Input:
Course codes: "CS101", "MA101", "CS101"
Expected Output:
Registered Courses: CS101, MA101

```java
import java.util.*;

public class CourseReg {
    public static void main(String[] args) {
        Set<String> courses = new LinkedHashSet<>();
        courses.add("CS101");
        courses.add("MA101");
        courses.add("CS101");
        System.out.println("Registered Courses: " + String.join(", ", courses));
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 5> javac CourseReg.java
PS C:\Users\baenu\Test\OOPJ Assignment 5> java CourseReg
Registered Courses: CS101, MA101
```

Problem 9: Event Attendance Counter
Use Case: Count unique attendees at an event, handling duplicate check-ins.
Requirements:
- Record attendee names
- Count only unique attendees
- Handle duplicate entries

Sample Input:
Attendees: "Amit", "Rohan", "Amit", "Priya"
Expected Output:
Total unique attendees: 3

```java
import java.util.*;

public class AttendanceCounter {
    public static void main(String[] args) {
        Set<String> attendees = new HashSet<>();
        String[] names = {"Amit", "Rohan", "Amit", "Priya"};
        attendees.addAll(Arrays.asList(names));
        System.out.println("Total unique attendees: " + attendees.size());
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 5> javac AttendanceCounter.java
PS C:\Users\baenu\Test\OOPJ Assignment 5> java AttendanceCounter.java
Total unique attendees: 3
```

Problem 10: Electronic Voting System
Use Case: Track unique voters in an election system to prevent duplicate voting.
Requirements:
● Record voter IDs
● Ensure one vote per voter
● Count total unique voters

Sample Input:
Voter IDs: 201, 202, 203, 202
Expected Output:
Total voters: 3

```java
import java.util.*;

public class VotingSystem {
    public static void main(String[] args) {
        Set<Integer> voters = new HashSet<>();
        int[] ids = {201, 202, 203, 202};
        for (int id : ids) voters.add(id);
        System.out.println("Total voters: " + voters.size());
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 5> javac VotingSystem.java
PS C:\Users\baenu\Test\OOPJ Assignment 5> java VotingSystem
Total voters: 3
```

Problem 11: Student Grade Management
Use Case: A teacher needs to map student names to their exam marks.
Requirements:
● Store student name and marks pairs
● Retrieve marks by student name
● Display all student-marks mappings

Sample Input:
Students and marks: "Amit" → 85, "Priya" → 92, "Rohan" → 78
Expected Output:
Grade Report: Amit:85, Priya:92, Rohan:78

```java
import java.util.*;

public class GradeMgmt {
    public static void main(String[] args) {
        Map<String, Integer> grades = new HashMap<>();
        grades.put("Amit", 85);
        grades.put("Priya", 92);
        grades.put("Rohan", 78);
        System.out.print("Grade Report: ");
        grades.forEach((k,v) -> System.out.print(k + ":" + v + ", "));
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 5> javac GradeMgmt.java
PS C:\Users\baenu\Test\OOPJ Assignment 5> java GradeMgmt
Grade Report: Priya:92, Amit:85, Rohan:78,
```

Problem 12: Attendance Tracking System
Use Case: Track student attendance percentages in alphabetical order.
Requirements:
- Map student names to attendance percentages
- Maintain alphabetical order of students
- Display sorted attendance report

Sample Input:
Attendance: "Amit" → 90, "Rohan" → 85, "Priya" → 95
Expected Output:
Attendance Report: Amit:90, Priya:95, Rohan:85

```java
import java.util.*;

public class AttendanceTrack {
    public static void main(String[] args) {
        Map<String, Integer> attendance = new TreeMap<>();
        attendance.put("Amit", 90);
        attendance.put("Rohan", 85);
        attendance.put("Priya", 95);
        System.out.print("Attendance Report: ");
        attendance.forEach((k,v) -> System.out.print(k + ":" + v + ", "));
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 5> javac AttendanceTrack.java
PS C:\Users\baenu\Test\OOPJ Assignment 5> java AttendanceTrack
Attendance Report: Amit:90, Priya:95, Rohan:85,
```

Problem 13: Student Registration Order Tracker
Use Case: Maintain the order in which students registered for a course.
Requirements:
● Record registration order
● Map student names to roll numbers
● Preserve insertion order

Sample Input:
Registrations: "Amit" → 101, "Rohan" → 102, "Priya" → 103
Expected Output:
Registration Order: Amit:101, Rohan:102, Priya:103

```java
import java.util.*;

public class RegOrder {
    public static void main(String[] args) {
        Map<String, Integer> reg = new LinkedHashMap<>();
        reg.put("Amit", 101);
        reg.put("Rohan", 102);
        reg.put("Priya", 103);
        System.out.print("Registration Order: ");
        reg.forEach((k,v) -> System.out.print(k + ":" + v + ", "));
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 5> javac RegOrder.java
PS C:\Users\baenu\Test\OOPJ Assignment 5> java RegOrder
Registration Order: Amit:101, Rohan:102, Priya:103,
```

Problem 14: Grade Update System
Use Case: Update a student's marks in the grading system.
Requirements:
● Store student grades
● Update existing student's marks
● Display updated information

Sample Input:
Initial: "Rohan" → 78
Update: "Rohan" → 88
Expected Output:
Updated Grade: Rohan:88

```java
import java.util.*;

public class GradeUpdate {
    public static void main(String[] args) {
        Map<String, Integer> grades = new HashMap<>();
        grades.put("Rohan", 78);
        grades.put("Rohan", 88);
        System.out.println("Updated Grade: Rohan:" + grades.get("Rohan"));
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 5> javac GradeUpdate.java
PS C:\Users\baenu\Test\OOPJ Assignment 5> java GradeUpdate
Updated Grade: Rohan:88
```

Problem 15: Library Book Inventory
Use Case: Track available copies of books in a library system.
Requirements:
- Map book titles to available copies
- Update copies when books are borrowed
- Display current inventory

Sample Input:
Initial inventory: "Java" → 3, "Python" → 5
Borrow: "Java" (1 copy)
Expected Output:
Current inventory: Java:2, Python:5

```java
import java.util.*;

public class BookInventory {
    public static void main(String[] args) {
        Map<String, Integer> books = new HashMap<>();
        books.put("Java", 3);
        books.put("Python", 5);
        books.put("Java", books.get("Java") - 1);
        System.out.print("Current inventory: ");
        books.forEach((k,v) -> System.out.print(k + ":" + v + ", "));
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 5> javac BookInventory.java
PS C:\Users\baenu\Test\OOPJ Assignment 5> java BookInventory
Current inventory: Java:2, Python:5,
```

Problem 16: Grade-Based Student Filter
Use Case: Remove students with failing grades from the honor roll.
Requirements:
● Store student grades in a map
● Remove students with marks below 60
● Display remaining student

Sample Input:
Student grades: "Amit" → 85, "Priya" → 52, "Rohan" → 78
Filter threshold: 60
Expected Output:
Honor Roll: Amit:85, Rohan:78

```java
import java.util.*;

public class GradeFilter {
    public static void main(String[] args) {
        Map<String,Integer> grades = new HashMap<>();
        grades.put("Amit",85);
        grades.put("Priya",52);
        grades.put("Rohan",78);
        grades.values().removeIf(v -> v < 60);
        System.out.print("Honor Roll: ");
        grades.forEach((k,v) -> System.out.print(k + ":" + v + ", "));
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 5> javac GradeFilter.java
PS C:\Users\baenu\Test\OOPJ Assignment 5> java GradeFilter
Honor Roll: Amit:85, Rohan:78,
```

Problem 17: Grade Distribution Counter
Use Case: Analyze the distribution of grades in a class.
Requirements:
● Count frequency of each grade
● Display grade distribution
● Handle multiple occurrences

Sample Input:
Grades: ["A", "B", "A", "C", "B", "A"]
Expected Output:
Grade Distribution: A=3, B=2, C=1

```java
import java.util.*;

public class GradeDist {
    public static void main(String[] args) {
        String[] arr = {"A","B","A","C","B","A"};
        Map<String,Integer> dist = new HashMap<>();
        for(String g:arr) dist.put(g, dist.getOrDefault(g,0)+1);
        System.out.print("Grade Distribution: ");
        dist.forEach((k,v)->System.out.print(k+"="+v+", "));
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 5> javac GradeDist.java
PS C:\Users\baenu\Test\OOPJ Assignment 5> java GradeDist
Grade Distribution: A=3, B=2, C=1,
```

Problem 18: Batch Merger System
Use Case: Merge student lists from morning and evening batches, removing duplicates.
Requirements:
- Combine two student lists
- Remove duplicate students
- Maintain unique student list

Sample Input:
Morning batch: "Amit", "Priya"
Evening batch: "Rohan", "Priya"
Expected Output:
Combined batches: Amit, Priya, Rohan

```java
import java.util.*;

public class BatchMerge {
    public static void main(String[] args) {
        Set<String> students = new LinkedHashSet<>();
        students.addAll(Arrays.asList("Amit","Priya"));
        students.addAll(Arrays.asList("Rohan","Priya"));
        System.out.println("Combined batches: " + String.join(", ", students));
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 5> javac BatchMerge.java
PS C:\Users\baenu\Test\OOPJ Assignment 5> java BatchMerge
Combined batches: Amit, Priya, Rohan
```

Problem 19: Grade Report Generator
Use Case: Display all student grades using proper iteration techniques.
Requirements:
● Iterate through student-grade mappings
● Display formatted grade report
● Use Iterator pattern

Sample Input:
Student grades: "Amit" → 85, "Priya" → 92
Expected Output:
Grade Report: Amit:85, Priya:92

```java
import java.util.*;

public class GradeReport {
    public static void main(String[] args) {
        Map<String,Integer> grades = new HashMap<>();
        grades.put("Amit",85);
        grades.put("Priya",92);
        Iterator<Map.Entry<String,Integer>> it = grades.entrySet().iterator();
        System.out.print("Grade Report: ");
        while(it.hasNext()){
            Map.Entry<String,Integer> e = it.next();
            System.out.print(e.getKey()+":"+e.getValue());
            if(it.hasNext()) System.out.print(", ");
        }
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 5> javac GradeReport.java
PS C:\Users\baenu\Test\OOPJ Assignment 5> java GradeReport
Grade Report: Priya:92, Amit:85
```

Problem 20: Even Roll Number Filter
Use Case: Filter and display only students with even roll numbers.
Requirements:
● Process list of roll numbers
● Remove odd roll numbers
● Display filtered results

Sample Input:
Roll numbers: 101, 102, 103, 104
Expected Output:
Even Roll Numbers: 102, 104

```java
import java.util.*;

public class EvenRoll {
    public static void main(String[] args) {
        List<Integer> rolls = new ArrayList<>(Arrays.asList(101,102,103,104));
        rolls.removeIf(n -> n%2!=0);
        System.out.println("Even Roll Numbers: " + rolls);
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 5> javac EvenRoll.java
PS C:\Users\baenu\Test\OOPJ Assignment 5> java EvenRoll
Even Roll Numbers: [102, 104]
```

Problem 21: Text Editor Undo System
Use Case: Implement undo functionality for a text editor.
Requirements:
- Track user actions in order
- Implement undo operation (LIFO)
- Display current action history

Sample Input:
Actions: "Type A", "Type B", "Delete"
Operation: Undo last action
Expected Output:
Current actions: Type A, Type B

```java
import java.util.*;

public class UndoSystem {
    public static void main(String[] args) {
        Stack<String> actions = new Stack<>();
        actions.push("Type A");
        actions.push("Type B");
        actions.push("Delete");
        actions.pop();
        System.out.println("Current actions: " + String.join(", ", actions));
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 5> javac UndoSystem.java
PS C:\Users\baenu\Test\OOPJ Assignment 5> java UndoSystem
Current actions: Type A, Type B
```

Problem 22: Ticket Booking Queue
Use Case: Manage customer service in a ticket booking system.
Requirements:
● Queue customers for service
● Serve customers in FIFO order
● Display current queue status

Sample Input:
Queue: "Amit", "Priya", "Rohan"
Serve: 1 customer
Expected Output:
Serving: Amit, Queue: Priya, Rohan

```java
import java.util.*;

public class TicketQueue {
    public static void main(String[] args) {
        Queue<String> q = new LinkedList<>();
        q.add("Amit");
        q.add("Priya");
        q.add("Rohan");
        String serve = q.poll();
        System.out.println("Serving: " + serve + ", Queue: " + String.join(", ", q));
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 5> javac TicketQueue.java
PS C:\Users\baenu\Test\OOPJ Assignment 5> java TicketQueue
Serving: Amit, Queue: Priya, Rohan
```

Problem 23: Browser History Management
Use Case: Maintain browser history with back functionality.
Requirements:
● Store visited pages
● Implement back navigation (LIFO)
● Display current history

Sample Input:
Pages visited: "Google", "YouTube", "GFG"
Action: Back (1 page)
Expected Output:
Current history: Google, YouTube

```java
import java.util.*;

public class BrowserHistory {
    public static void main(String[] args) {
        Stack<String> history = new Stack<>();
        history.push("Google");
        history.push("YouTube");
        history.push("GFG");
        history.pop();
        System.out.println("Current history: " + String.join(", ", history));
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 5> javac BrowserHistory.java
PS C:\Users\baenu\Test\OOPJ Assignment 5> java BrowserHistory
Current history: Google, YouTube
```

Problem 24: Print Job Queue Manager
Use Case: Manage print jobs in a shared printer system.
Requirements:
- Queue print jobs
- Process jobs in order
- Display job status

Sample Input:
Jobs: "Doc1", "Doc2", "Doc3"
Process: 1 job
Expected Output:
Printing Doc1, Queue: Doc2, Doc3

```java
import java.util.*;

public class PrintQueue {
    public static void main(String[] args) {
        Queue<String> jobs = new LinkedList<>();
        jobs.add("Doc1");
        jobs.add("Doc2");
        jobs.add("Doc3");
        String job = jobs.poll();
        System.out.println("Printing " + job + ", Queue: " + String.join(", ", jobs));
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 5> javac PrintQueue.java
PS C:\Users\baenu\Test\OOPJ Assignment 5> java PrintQueue
Printing Doc1, Queue: Doc2, Doc3
```

Problem 25: Command History Tracker
Use Case: Store recent commands in a terminal with limited history.
Requirements:
● Maintain last 3 commands
● Remove oldest when limit exceeded
● Display recent commands

Sample Input:
Commands: "ls", "pwd", "cd ..", "mkdir"
History limit: 3
Expected Output:
Recent Commands: pwd, cd .., mkdir

```java
import java.util.*;

public class CmdHistory {
    public static void main(String[] args) {
        Queue<String> cmds = new LinkedList<>();
        String[] input = {"ls","pwd","cd ..","mkdir"};
        for(String c:input){
            if(cmds.size()==3) cmds.poll();
            cmds.add(c);
        }
        System.out.println("Recent Commands: " + String.join(", ", cmds));
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 5> javac CmdHistory.java
PS C:\Users\baenu\Test\OOPJ Assignment 5> java CmdHistory
Recent Commands: pwd, cd .., mkdir
```

Problem 26: Employee Management System
Use Case: Manage employee information including name and salary.
Requirements:
● Create Employee objects with name and salary
● Store employees in a collection
● Display employee information

Sample Input:
Employees: Employee("Amit", 50000), Employee("Priya", 60000)
Expected Output:
Employee List: Amit:50000, Priya:60000

```java
import java.util.*;

class Employee {
    String name;
    int salary;
    Employee(String n,int s){name=n;salary=s;}
    public String toString(){return name+":"+salary;}
}

public class EmpMgmt {
    public static void main(String[] args) {
        List<Employee> emps = new ArrayList<>();
        emps.add(new Employee("Amit",50000));
        emps.add(new Employee("Priya",60000));
        System.out.print("Employee List: ");
        for(Employee e:emps) System.out.print(e+", ");
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 5> javac EmpMgmt.java
PS C:\Users\baenu\Test\OOPJ Assignment 5> java EmpMgmt
Employee List: Amit:50000, Priya:60000,
```

Problem 27: Employee Salary Sorting
Use Case: Sort employees by salary for payroll processing.
Requirements:
- Store employee objects
- Sort by salary in ascending order
- Display sorted employee list

Sample Input:
Employees: ("Amit", 50000), ("Priya", 60000), ("Rohan", 45000)
Expected Output:
Sorted by salary: Rohan:45000, Amit:50000, Priya:60000

```java
import java.util.*;

class Emp {
    String name;
    int salary;
    Emp(String n,int s){name=n;salary=s;}
    public String toString(){return name+":"+salary;}
}

public class EmpSort {
    public static void main(String[] args) {
        List<Emp> list = new ArrayList<>();
        list.add(new Emp("Amit",50000));
        list.add(new Emp("Priya",60000));
        list.add(new Emp("Rohan",45000));
        list.sort(Comparator.comparingInt(e->e.salary));
        System.out.print("Sorted by salary: ");
        for(Emp e:list) System.out.print(e+", ");
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 5> javac EmpSort.java
PS C:\Users\baenu\Test\OOPJ Assignment 5> java EmpSort
Sorted by salary: Rohan:45000, Amit:50000, Priya:60000,
```

Problem 28: Department Employee Mapping
Use Case: Organize employees by department for HR management.
Requirements:
- Map departments to employee lists
- Group employees by department
- Display departmental structure

Sample Input:
Department mapping: "IT" → ["Amit", "Rohan"], "HR" → ["Priya"]
Expected Output:
Department Structure: IT: Amit, Rohan; HR: Priya

```java
import java.util.*;

public class DeptMap {
    public static void main(String[] args) {
        Map<String,List<String>> dept = new HashMap<>();
        dept.put("IT", Arrays.asList("Amit","Rohan"));
        dept.put("HR", Arrays.asList("Priya"));
        System.out.print("Department Structure: ");
        dept.forEach((k,v)->System.out.print(k+": "+String.join(", ",v)+"; "));
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 5> javac DeptMap.java
PS C:\Users\baenu\Test\OOPJ Assignment 5> java DeptMap
Department Structure: HR: Priya; IT: Amit, Rohan;
```

Problem 29: Student Record System
Use Case: Maintain student records with name and grade information.
Requirements:
- Create Student objects with name and grade
- Store in a collection
- Display student records

Sample Input:
Students: Student("Amit", "A"), Student("Priya", "B")
Expected Output:
Student Records: Amit:A, Priya:B

```java
import java.util.*;

class Student {
    String name, grade;
    Student(String n,String g){name=n;grade=g;}
    public String toString(){return name+":"+grade;}
}

public class StudRecord {
    public static void main(String[] args) {
        List<Student> list = new ArrayList<>();
        list.add(new Student("Amit","A"));
        list.add(new Student("Priya","B"));
        System.out.print("Student Records: ");
        for(Student s:list) System.out.print(s+", ");
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 5> javac StudRecord.java
PS C:\Users\baenu\Test\OOPJ Assignment 5> java StudRecord
Student Records: Amit:A, Priya:B,
```

Problem 30: Grade-Based Student Filter
Use Case: Filter students based on minimum grade requirements.
Requirements:
- Store student objects with grades
- Remove students below grade B
- Display filtered results

Sample Input:
Students: ("Amit", "A"), ("Priya", "C"), ("Rohan", "B")
Filter: Grade >= B
Expected Output:
Qualified Students: Amit:A, Rohan:B

```java
class Stud {
    String name, grade;
    Stud(String n,String g){name=n;grade=g;}
    public String toString(){return name+":"+grade;}
}

public class StudGradeFilter {
    public static void main(String[] args) {
        List<Stud> list = new ArrayList<>();
        list.add(new Stud("Amit","A"));
        list.add(new Stud("Priya","C"));
        list.add(new Stud("Rohan","B"));
        list.removeIf(s -> s.grade.compareTo("B")>0);
        System.out.print("Qualified Students: ");
        for(Stud s:list) System.out.print(s+", ");
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 5> javac StudGradeFilter.java
PS C:\Users\baenu\Test\OOPJ Assignment 5> java StudGradeFilter
Qualified Students: Amit:A, Rohan:B,
```

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)

Requirements
1. Registration List - ArrayList<Student>
- Store students in registration order

2. Merit List - Comparable<Student>
- Sort by CGPA (descending order)

3. Alphabetical List - Comparator<Student>
- Sort by name (A to Z)

4. Department Grouping - HashMap<String, List<Student>>
- Group students by department

5. Unique Names - HashSet<String>
- Track unique student names

6. Roll Number Sorting - TreeSet<Student>
- Auto-sort by roll number

7. Performance Filter - Iterator
- Remove students with CGPA < 5.0

8. Recent Registrations - Stack<Student>
- Track last added students (LIFO)

9. Scholarship Queue - Queue<Student>
- Process students for scholarships (FIFO)

10. Hostel Applications - LinkedList<Integer>
- Add priority applicants at **front**
- Add regular applicants at **end**
- Remove from **both ends** for allocation

Sample Data
```
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);
```

Expected Outputs
Registration Order: Amit, Priya, Rohan, Sneha
Merit List: Priya(9.2), Amit(8.5), Rohan(7.8), Sneha(4.5)
Alphabetical: Amit, Priya, Rohan, Sneha
Department Groups:
CS: [Amit, Rohan]
Math: [Priya]
Physics: [Sneha]
After Filter (CGPA >= 5.0): Amit, Priya, Rohan
Hostel Queue:
Add regular(105): [105]
Add priority(101): [101, 105]
Remove front: [105]
CONGRATULATIONS

```java
import java.util.*;

class Student implements Comparable<Student> {
    int rollNumber;
    String name;
    String department;
    double cgpa;

    public Student(int rollNumber, String name, String department, double cgpa) {
        this.rollNumber = rollNumber;
        this.name = name;
        this.department = department;
        this.cgpa = cgpa;
    }

    @Override
    public int compareTo(Student other) {
        return Double.compare(other.cgpa, this.cgpa);
    }

    @Override
    public int hashCode() {
        return Objects.hash(rollNumber);
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (!(obj instanceof Student)) return false;
        Student other = (Student) obj;
        return this.rollNumber == other.rollNumber;
    }

    @Override
    public String toString() {
        return name + "(" + cgpa + ")";
    }
}
```

```java
public class UniversityStudentManagement {
    public static void main(String[] args) {
        //Data
        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<Student> registrationList = new ArrayList<>();
        registrationList.add(s1);
        registrationList.add(s2);
        registrationList.add(s3);
        registrationList.add(s4);

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

        //Merit
        List<Student> meritList = new ArrayList<>(registrationList);
        Collections.sort(meritList);
        System.out.println("Merit List: " + meritList);

        //Alphabetical
        List<Student> alphabeticalList = new ArrayList<>(registrationList);
        alphabeticalList.sort(Comparator.comparing(st -> st.name));
        System.out.println("Alphabetical: " + alphabeticalList);

        //Department grouping
        Map<String, List<Student>> departmentGroups = new HashMap<>();
        for (Student st : registrationList) {
            departmentGroups.computeIfAbsent(st.department, k -> new ArrayList<>()).add(st);
        }
        System.out.println("Department Groups: " + departmentGroups);

        //Unique names
        Set<String> uniqueNames = new HashSet<>();
        for (Student st : registrationList) {
            uniqueNames.add(st.name);
        }
        System.out.println("Unique Names: " + uniqueNames);

        //Roll number sorting
        Set<Student> rollNumberSet = new TreeSet<>(Comparator.comparingInt(st -> st.rollNumber));
        rollNumberSet.addAll(registrationList);
        System.out.println("Roll Number Sorted: " + rollNumberSet);
```

```java
        //Performance filter
        Iterator<Student> it = registrationList.iterator();
        while (it.hasNext()) {
            if (it.next().cgpa < 5.0) {
                it.remove();
            }
        }
        System.out.println("After Filter (CGPA >= 5.0): " + registrationList);

        //Recent registrations
        Stack<Student> stack = new Stack<>();
        stack.push(s1);
        stack.push(s2);
        stack.push(s3);
        stack.push(s4);
        System.out.println("Stack (Recent Registrations): " + stack);
        System.out.println("Popped from Stack: " + stack.pop());

        //Scholarship queue
        Queue<Student> scholarshipQueue = new LinkedList<>();
        scholarshipQueue.offer(s1);
        scholarshipQueue.offer(s2);
        scholarshipQueue.offer(s3);
        System.out.println("Scholarship Queue: " + scholarshipQueue);
        System.out.println("Polled from Queue: " + scholarshipQueue.poll());

        //Hostel applications
        LinkedList<Integer> hostelQueue = new LinkedList<>();
        hostelQueue.addLast(105); // regular
        System.out.println("Add regular(105): " + hostelQueue);
        hostelQueue.addFirst(101); // priority
        System.out.println("Add priority(101): " + hostelQueue);
        hostelQueue.removeFirst();
        System.out.println("Remove front: " + hostelQueue);
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 5> javac UniversityStudentManagement.java
PS C:\Users\baenu\Test\OOPJ Assignment 5> java UniversityStudentManagement
Registration Order: [Amit(8.5), Priya(9.2), Rohan(7.8), Sneha(4.5)]
Merit List: [Priya(9.2), Amit(8.5), Rohan(7.8), Sneha(4.5)]
Alphabetical: [Amit(8.5), Priya(9.2), Rohan(7.8), Sneha(4.5)]
Department Groups: {CS=[Amit(8.5), Rohan(7.8)], Math=[Priya(9.2)], Physics=[Sneha(4.5)]}
Unique Names: [Priya, Sneha, Amit, Rohan]
Roll Number Sorted: [Amit(8.5), Priya(9.2), Rohan(7.8), Sneha(4.5)]
After Filter (CGPA >= 5.0): [Amit(8.5), Priya(9.2), Rohan(7.8)]
Stack (Recent Registrations): [Amit(8.5), Priya(9.2), Rohan(7.8), Sneha(4.5)]
Popped from Stack: Sneha(4.5)
Scholarship Queue: [Amit(8.5), Priya(9.2), Rohan(7.8)]
Polled from Queue: Amit(8.5)
Add regular(105): [105]
Add priority(101): [101, 105]
Remove front: [105]
```