

1. Care sunt, în codul de pe verso<sup>(\*)</sup>, clasele, clasele abstracte, interfețele și clasele interne? (definite sau doar utilizate în cod)
2. Ce este suprascrierea? Dar supraîncărcarea? Recunoașteți câte un exemplu pentru fiecare din codul de pe verso.
3. Cum sunt reprezentate relațiile dintre obiectele de tip Airline și cum sunt folosite?
4. La rulare, programul arunca excepția de mai jos. Corectați-l.

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: -1
    at airline.AirlineProblem.canRedeem(AirlineProblem.java:79)
    at airline.AirlineProblem.main(AirlineProblem.java:38)
```

5. Ce înțelegeți prin încapsulare? Găsiți și explicați un exemplu în codul de pe verso.

<sup>(\*)</sup> codul este inspirat din exemplul de la <https://www.cs.utexas.edu/~scottm/cs307/codingSamples.htm>

```

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class AirlineProblem {

    public static void main(String[] args) {
        Scanner scannerToReadAirlines = null;
        try {
            scannerToReadAirlines = new Scanner(new File("input.txt"));
        } catch (IOException e) {
            System.out.println("Could not connect to file input.txt.");
            System.exit(0);
        }
        if (scannerToReadAirlines != null) {
            List<Airline> airlinesPartnersNetwork = new ArrayList<Airline>();
            Airline newAirline;
            String lineFromFile;
            String[] airlineNames;
            while (scannerToReadAirlines.hasNext()) {
                lineFromFile = scannerToReadAirlines.nextLine();
                airlineNames = lineFromFile.split(",");
                newAirline = new Airline(airlineNames);
                airlinesPartnersNetwork.add(newAirline);
            }
            System.out.println(airlinesPartnersNetwork);
            Scanner keyboard = new Scanner(System.in);
            System.out.print("Enter airline miles are on: ");
            String start = keyboard.nextLine();
            System.out.print("Enter goal airline: ");
            String goal = keyboard.nextLine();
            List<String> pathForMiles = new ArrayList<String>();
            List<String> airlinesVisited = new ArrayList<String>();
            if (canRedeem(start, goal, pathForMiles, airlinesVisited,
                airlinesPartnersNetwork))
                System.out.println("Path to redeem miles: " + pathForMiles);
            else
                System.out.println("Cannot convert miles from " + start + " to "
                    + goal + ".");
        }
    }

    private static boolean canRedeem(String current, String goal,
        List<String> pathForMiles, List<String> airlinesVisited,
        List<Airline> network) {
        if (current.equals(goal)) {
            // base case 1, I have found a path!
            pathForMiles.add(current);
            return true;
        } else if (airlinesVisited.contains(current))
            // base case 2, I have already been here, don't go into a cycle
            return false;
        else {
            // I have not been here and it isn't the goal
            // so check its partners; now I have been here
            airlinesVisited.add(current);

            // add this to the path
            pathForMiles.add(current);

```

```

// find this airline in the network
int pos = -1;
int index = 0;
while (pos == -1 && index < network.size()) {
    if (network.get(index).getName().equals(current))
        pos = index;
    index++;
}
// if not in the network, no partners
if (pos == -1)
    return false;

// loop through partners
index = -1;
String[] partners = network.get(pos).getPartners();
boolean foundPath = false;
while (!foundPath && index < partners.length) {
    foundPath = canRedeem(partners[index], goal, pathForMiles,
        airlinesVisited, network);

    index++;
}
if (!foundPath)
    pathForMiles.remove(pathForMiles.size() - 1);
return foundPath;
}

private static class Airline {
    private String name;
    private ArrayList<String> partners;

    public Airline(String[] data) {
        assert data != null && data.length > 0 : "Failed precondition";
        name = data[0];
        partners = new ArrayList<String>();
        for (int i = 1; i < data.length; i++)
            partners.add(data[i]);
    }

    public String[] getPartners() {
        return partners.toArray(new String[partners.size()]);
    }

    public boolean isPartner(String name) {
        return partners.contains(name);
    }

    public boolean isPartner(Airline airline) {
        return partners.contains(airline);
    }

    public String getName() {
        return name;
    }

    @Override
    public String toString() {
        return name + ", partners: " + partners;
    }
}
}

```

1. Care sunt, în codul de pe verso<sup>(\*)</sup>, clasele, clasele abstracte, interfețele și clasele interne? (definite sau doar utilizate în cod)
2. Ce este suprascrierea? Dar supraîncărcarea? Recunoașteți câte un exemplu pentru fiecare din codul de pe verso.
3. Ce este upcast? Dar downcast? Dați câte un exemplu pentru fiecare bazat pe variabilele din codul de pe verso.
4. Știind că descrierea metodei *poll* din clasa *PriorityQueue* este *“Retrieves and removes the head of this queue, or returns null if this queue is empty.”*, ce se va afișa în urma rulării codului? Cum ar trebui modificat codul pentru a oferi o funcționare mai corectă, care să nu afișeze valori *null*?
5. Ce înțelegeți prin polimorfism? Găsiți și explicați un exemplu în codul de pe verso.

<sup>(\*)</sup> codul este inspirat din exemplul de la <http://javaconceptsoftheday.com/java-priorityqueue-example/>

```

class Employee {

    private String name;
    private int hoursInOffice;
    private int tasksDone;

    public Employee(String name, int hoursInOffice, int tasksDone) {
        super();
        this.name = name;
        this.hoursInOffice = hoursInOffice;
        this.tasksDone = tasksDone;
    }

    public int getHoursInOffice() {
        return hoursInOffice;
    }

    public int getTasksDone() {
        return tasksDone;
    }

    @Override
    public String toString() {
        return name + " (hoursInOffice=" + hoursInOffice +
            ", tasksDone=" + tasksDone + ")";
    }

}

interface EmployeeComparator extends Comparator<Employee> {
}

class SuperficialComparator implements EmployeeComparator {

    @Override
    public int compare(Employee o1, Employee o2) {
        return o2.getHoursInOffice() - o1.getHoursInOffice();
    }

}

class AnalyticalComparator implements EmployeeComparator {

    @Override
    public int compare(Employee o1, Employee o2) {
        return o2.getTasksDone() - o1.getTasksDone();
    }

}

```

```

public class Manager {

    private String name;
    private PriorityQueue<Employee> evaluation;

    public Manager(String name, EmployeeComparator comparator) {
        this.name = name;
        evaluation = new PriorityQueue<Employee>(10, comparator);
    }

    public void hire(Employee e) {
        evaluation.offer(e);
    }

    public void getTopEmployees(int count) {
        for (int i = 0; i < count; i++)
            System.out.println(evaluation.poll());
    }

    public void getTopEmployees() {
        getTopEmployees(4);
    }

}

public class PriorityQueueExample {

    public static void main(String[] args) {
        Employee e1 = new Employee("Andrei", 40, 5);
        Employee e2 = new Employee("Marius", 60, 15);
        Employee e3 = new Employee("Daniel", 30, 25);

        Manager superficialMgr = new Manager("dl Superficial",
            new SuperficialComparator());
        superficialMgr.hire(e1);
        superficialMgr.hire(e2);
        superficialMgr.hire(e3);
        superficialMgr.getTopEmployees();
        System.out.println();
        Manager analyticalMgr = new Manager("dl Analytic",
            new AnalyticalComparator());
        analyticalMgr.hire(e1);
        analyticalMgr.hire(e2);
        analyticalMgr.hire(e3);
        analyticalMgr.getTopEmployees();
    }

}

```

1. Care sunt, în codul de pe verso<sup>(\*)</sup>, clasele, clasele abstracte, interfețele și clasele interne? (definite sau doar utilizate în cod)
2. Ce este upcast? Dar downcast? Dați câte un exemplu pentru fiecare bazat pe variabilele din codul de pe verso.
3. Ce face cuvântul cheie static? Discutați despre ideea de a îl folosi pentru metoda sort.
4. La rulare, programul arunca excepția de mai jos. Corectați-l:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 11
    at Sorter.quickSort(Sorter.java:33)
    at Sorter.sort(Sorter.java:13)
    at Program.main(Program.java:8)
```

5. Ce înțelegeți prin agregare? Găsiți și explicați un exemplu în codul de pe verso.

<sup>(\*)</sup> codul este inspirat din exemplul de la <http://www.java2novice.com/java-sorting-algorithms/quick-sort>

```

public class Sorter {

    private int array[];
    private int length;

    public void sort(int[] inputArr) {

        if (inputArr == null || inputArr.length == 0) {
            return;
        }
        this.array = inputArr;
        length = inputArr.length;
        quickSort(0, length);
    }

    public void quickSort(int lowerIndex, int higherIndex) {

        int i = lowerIndex;
        int j = higherIndex;
        // calculate pivot number, taking pivot as middle index no.
        int pivot = array[lowerIndex + (higherIndex-lowerIndex) / 2];
        // Divide into two arrays
        while (i <= j) {
            /**
             * In each iteration, identify a number from left side which
             * is greater then the pivot value, and also identify a
             * number from right side which is less then the pivot value.
             * Once the search is done, then we exchange both numbers.
             */
            while (array[i] < pivot) {
                i++;
            }
            while (array[j] > pivot) {
                j--;
            }
            if (i <= j) {
                exchangeNumbers(i, j);
                // move index to next position on both sides
                i++;
                j--;
            }
        }
        // call quickSort() method recursively
        if (lowerIndex < j)
            quickSort(lowerIndex, j);
        if (i < higherIndex)
            quickSort(i, higherIndex);
    }
}

```

```

        private void exchangeNumbers(int i, int j) {
            int temp = array[i];
            array[i] = array[j];
            array[j] = temp;
        }
    }

    public class Program {

        public static void main(String a[]) {

            Sorter sorter = new Sorter();
            int[] input = { 24, 2, 45, 20, 56, 75, 2, 56, 99, 53, 12 };
            sorter.sort(input);
            for (int i : input) {
                System.out.print(i);
                System.out.print(" ");
            }
        }
    }

    public abstract class Student {

        protected String name;
        protected String level;
    }

    public class UniversityStudent extends Student {

        public UniversityStudent(String name) {
            this.name = name;
            this.level = "university";
        }
    }

    public class HighSchoolStudent extends Student {

        public HighSchoolStudent(String name) {
            this.name = name;
            this.level = "high-school";
        }
    }
}

```