**Problem statement**

Shyam is working on a set of employee records in order to find the senior people based on their age. Help him sort out the papers using the selection sort technique.

**Input format :**

The first line consists of an integer **N,**representing the number of dates.

The following **N** lines consist of the respective dates as space-separated integers in **dd mm yyyy** format.

**Output format :**

The output displays the sorted order of the records based on the date.

**Refer to the sample output for the formatting specifications.**

**Code constraints :**

1 ≤ N ≤ 10

**Sample test cases :**

**Input 1 :**

5

20 04 2016

15 06 2018

11 09 2019

30 04 1992

16 11 2000

**Output 1 :**

30 4 1992

16 11 2000

20 4 2016

15 6 2018

11 9 2019

**Solution**:

import java.util.Scanner;

class DateSorter {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in); // Create a Scanner object to read input from the console

        int n = scanner.nextInt(); // Read the number of dates to be sorted

        scanner.nextLine(); // Consume the newline character left by nextInt()

        String[] dates = new String[n]; // Create an array to store the dates

        for (int i = 0; i < n; i++) {

            dates[i] = scanner.nextLine(); // Read each date from the input and store it in the array

        }

        selectionSort(dates); // Sort the array of dates using the selection sort algorithm

        for (String date : dates) {

            System.out.println(formatDate(date)); // Print each date in the sorted order

        }

        scanner.close(); // Close the scanner

    }

    // Function to sort an array of dates using the selection sort algorithm

    public static void selectionSort(String[] dates) {

        int n = dates.length; // Get the length of the array

        for (int i = 0; i < n - 1; i++) { // Iterate over the array elements

            int minIndex = i; // Assume the current element is the minimum

            for (int j = i + 1; j < n; j++) { // Iterate over the unsorted part of the array

                if (compareDates(dates[j], dates[minIndex]) < 0) { // Compare dates

                    minIndex = j; // Update minIndex if a smaller date is found

                }

            }

            // Swap the found minimum element with the current element

            String temp = dates[minIndex];

            dates[minIndex] = dates[i];

            dates[i] = temp;

        }

    }

    // Function to compare two dates

    public static int compareDates(String date1, String date2) {

        String[] parts1 = date1.split(" "); // Split the first date into day, month, and year

        String[] parts2 = date2.split(" "); // Split the second date into day, month, and year

        int day1 = Integer.parseInt(parts1[0]); // Convert day part of the first date to integer

        int month1 = Integer.parseInt(parts1[1]); // Convert month part of the first date to integer

        int year1 = Integer.parseInt(parts1[2]); // Convert year part of the first date to integer

        int day2 = Integer.parseInt(parts2[0]); // Convert day part of the second date to integer

        int month2 = Integer.parseInt(parts2[1]); // Convert month part of the second date to integer

        int year2 = Integer.parseInt(parts2[2]); // Convert year part of the second date to integer

        if (year1 != year2) { // Compare years

            return year1 - year2; // Return the difference if years are not equal

        } else if (month1 != month2) { // Compare months if years are equal

            return month1 - month2; // Return the difference if months are not equal

        } else { // Compare days if years and months are equal

            return day1 - day2; // Return the difference in days

        }

    }

    // Function to format the date as "day month year"

    public static String formatDate(String date) {

        String[] parts = date.split(" "); // Split the date into day, month, and year

        int day = Integer.parseInt(parts[0]); // Convert day part to integer

        int month = Integer.parseInt(parts[1]); // Convert month part to integer

        int year = Integer.parseInt(parts[2]); // Convert year part to integer

        return day + " " + month + " " + year; // Return the formatted date string

    }

}

**Explanation**

1. **Splitting the Dates**:
   * The dates are provided as strings in the format "day month year" (e.g., "25 12 2020").
   * split(" ") is used to divide the date string into parts: parts1 and parts2 arrays will hold the day, month, and year as separate strings.
2. **Converting Strings to Integers**:
   * Integer.parseInt(parts1[0]) converts the day part of the first date to an integer.
   * Integer.parseInt(parts1[1]) converts the month part of the first date to an integer.
   * Integer.parseInt(parts1[2]) converts the year part of the first date to an integer.
   * Similarly, the same conversions are done for the second date.
3. **Comparing Years**:
   * If the years (year1 and year2) are different, the function returns the difference between the two years.
   * This allows for sorting based on the year primarily.
4. **Comparing Months**:
   * If the years are the same, the function then compares the months (month1 and month2).
   * If the months are different, the function returns the difference between the two months.
   * This allows for sorting based on the month if the years are the same.
5. **Comparing Days**:
   * If both the years and months are the same, the function finally compares the days (day1 and day2).
   * The function returns the difference between the two days.

**Return Value**

* The return value is an integer:
  + A positive value indicates that date1 is later than date2.
  + A negative value indicates that date1 is earlier than date2.
  + A zero value indicates that the dates are the same.

**Problem Statement**

You are a teacher at a school, and you have recently conducted an exam for your students. The exam grades of the students are represented as an array of integers. You want to determine the number of inversions in the list of grades.

Given an array of integers, find the inversion count in the array using insertion sort.

**Inversion Count:** For an array, the inversion count indicates how far (or close) the array is from being sorted. If the array is already sorted, then the inversion count is 0. If an array is sorted in reverse order, then the inversion count is the maximum.

Formally, two elements arr[i] and arr[j] form an inversion if arr[i] > arr[j] and i < j.

**Example 1**

**Input:**

arr[] = {8, 4, 2, 1}

**Output:**

6

**Explanation:**

The sequence has six inversions (8, 4), (4, 2), (8, 2), (8, 1), (4, 1), (2, 1).

**Example 2**

**Input:**

arr[] = {29, 37, 42, 54, 61}

**Output:**

0

**Explanation**:

As the sequence is already sorted, there is no inversion count.

**Example 3**

**Input:**

arr[] = {100, 100, 100}

**Output:**

0

**Explanation**:

As all the elements of the array are the same, there is no inversion count.

**Note:** This question is asked in Amazon, Microsoft and Flipkart coding interviews.

**Input format :**

The first line of input consists of **N**, representing the size of the array.

The second line consists of the **N**array elements, separated by space.

**Output format :**

The output prints the inversion count in the given array.

**Refer to the sample output for formatting specifications**

**Code constraints :**

1 ≤ N ≤ 25

1 ≤ array elements ≤ 100

**Sample test cases :**

**Input 1 :**

4

8 4 2 1

**Output 1 :**

6

**Input 2 :**

5

20 45 19 37 58

**Output 2 :**

3

**Input 3 :**

5

29 37 42 54 61

**Output 3 :**

0

**Input 4 :**

3

100 100 100

**Output 4 :**

0

**Solution:**

import java.util.Scanner;

class InversionCount {

    // Method to count the number of inversions in the array using a modified insertion sort

    public static int countInversions(int[] arr) {

        int n = arr.length; // Get the length of the array

        int inversionCount = 0; // Initialize the inversion count to 0

        // Iterate through the array starting from the second element

        for (int i = 1; i < n; i++) {

            int key = arr[i]; // The current element to be compared

            int j = i - 1; // The index of the previous element

            // Move elements of arr[0..i-1], that are greater than key,

            // to one position ahead of their current position

            while (j >= 0 && arr[j] > key) {

                arr[j + 1] = arr[j]; // Shift the element to the right

                j--; // Move to the previous element

                inversionCount++; // Increment the inversion count

            }

            arr[j + 1] = key; // Place the key element at its correct position

        }

        return inversionCount; // Return the total number of inversions

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in); // Create a Scanner object to read input

        int n = scanner.nextInt(); // Read the number of elements in the array

        int[] arr = new int[n]; // Create an array to hold the elements

        // Read each element and store it in the array

        for (int i = 0; i < n; i++) {

            arr[i] = scanner.nextInt();

        }

        int result = countInversions(arr); // Call the countInversions method

        System.out.println(result); // Print the result (number of inversions)

        scanner.close(); // Close the scanner

    }

}

**Explanation**

1. **Imports and Class Definition**:
   * The java.util.Scanner class is imported to read input from the user.
   * The class InversionCount contains the logic for counting inversions in an array.
2. **countInversions Method**:
   * This method takes an integer array arr as input and returns the number of inversions in the array.
   * int n = arr.length; gets the length of the array.
   * int inversionCount = 0; initializes the inversion count to zero.
   * The method uses a modified insertion sort to count inversions:
     + For each element starting from the second element (i = 1 to i < n), the element (key) is compared to the elements before it (arr[j] where j = i - 1).
     + While the current element key is less than the previous elements (arr[j] > key), the previous elements are shifted one position to the right.
     + Each shift indicates an inversion, so inversionCount is incremented.
     + The key is then placed in its correct position (arr[j + 1] = key).
3. **main Method**:
   * The main method reads the array input from the user.
   * Scanner scanner = new Scanner(System.in); creates a Scanner object for input.
   * int n = scanner.nextInt(); reads the number of elements in the array.
   * int[] arr = new int[n]; creates an array to hold the input elements.
   * A for loop reads each element and stores it in the array.
   * int result = countInversions(arr); calls the countInversions method to count the inversions in the array.
   * System.out.println(result); prints the result.
   * scanner.close(); closes the scanner.

**Purpose**

* The purpose of this program is to count the number of inversions in an array. An inversion is a situation where a larger element precedes a smaller element in the array. This is useful in various computational problems, such as sorting and understanding the order or disorder within datasets.

**Problem Statement**

Kumar wants to design a program to analyze race timing data using the bubble sort algorithm. Write a program to help Kumar find and display the second-fastest race time in seconds from a given list of race times.

**Input format :**

The first line of input consists of an integer **N,** representing the number of race times.

The second line consists of **N** space-separated integers, representing the individual race times in seconds.

**Output format :**

If N is less than 2, print "Insufficient data to find the second-fastest time.".

Otherwise, print "X seconds" where X is the value of the second-fastest race time, in seconds.

**Refer to the sample output for the formatting specifications.**

**Code constraints :**

In this scenario, the given test cases will fall under the following constraints:

1 ≤ N ≤ 10

1 ≤ each race time ≤ 360

**Sample test cases :**

**Input 1 :**

5

34 21 56 12 94

**Output 1 :**

21 seconds

**Input 2 :**

6

115 124 234 345 159 360

**Output 2 :**

124 seconds

**Input 3 :**

1

298

**Output 3 :**

Insufficient data to find the second-fastest time.

**Input 4 :**

2

10 39

**Output 4 :**

39 seconds

**Solution:**

import java.util.Scanner;

class SecondFastestTime {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in); // Create a Scanner object to read input

        int n = scanner.nextInt(); // Read the number of race times

        // Check if there are fewer than 2 race times

        if (n < 2) {

            System.out.println("Insufficient data to find the second-fastest time.");

            return; // Exit the program if there are fewer than 2 race times

        }

        int[] raceTimes = new int[n]; // Create an array to hold the race times

        // Read each race time and store it in the array

        for (int i = 0; i < n; i++) {

            raceTimes[i] = scanner.nextInt();

        }

        bubbleSort(raceTimes); // Sort the race times using bubble sort

        // Print the second-fastest time (which is the second element in the sorted array)

        System.out.println(raceTimes[1] + " seconds");

        scanner.close(); // Close the scanner

    }

    // Method to sort an array using bubble sort

    public static void bubbleSort(int[] arr) {

        int n = arr.length; // Get the length of the array

        boolean swapped; // Variable to check if any swaps were made in a pass

        // Outer loop to control the number of passes

        for (int i = 0; i < n - 1; i++) {

            swapped = false; // Reset swapped to false for each pass

            // Inner loop to compare adjacent elements and swap if needed

            for (int j = 0; j < n - 1 - i; j++) {

                if (arr[j] > arr[j + 1]) {

                    // Swap arr[j] and arr[j + 1]

                    int temp = arr[j];

                    arr[j] = arr[j + 1];

                    arr[j + 1] = temp;

                    swapped = true; // Set swapped to true if a swap is made

                }

            }

            // If no swaps were made in this pass, the array is already sorted

            if (!swapped) break;

        }

    }

}

**Explanation**

1. **Imports and Class Definition**:
   * The java.util.Scanner class is imported to read input from the user.
   * The class SecondFastestTime contains the logic for finding the second-fastest race time.
2. **main Method**:
   * The main method is the entry point of the program.
   * Scanner scanner = new Scanner(System.in); creates a Scanner object for input.
   * int n = scanner.nextInt(); reads the number of race times.
   * **Check for Insufficient Data**:
     + If n is less than 2, the program prints "Insufficient data to find the second-fastest time." and exits. This check ensures that there are at least two times to find the second-fastest one.
   * **Reading Race Times**:
     + int[] raceTimes = new int[n]; creates an array to hold the race times.
     + A for loop reads each race time and stores it in the array.
   * **Sorting and Output**:
     + bubbleSort(raceTimes); calls the bubbleSort method to sort the race times.
     + System.out.println(raceTimes[1] + " seconds"); prints the second-fastest time, which is the second element in the sorted array.
   * scanner.close(); closes the scanner.
3. **bubbleSort Method**:
   * This method sorts the input array using the bubble sort algorithm.
   * int n = arr.length; gets the length of the array.
   * The outer loop runs n - 1 times, controlling the number of passes.
   * boolean swapped; is a flag to check if any swaps were made during a pass.
   * **Inner Loop and Swapping**:
     + The inner loop compares adjacent elements and swaps them if they are in the wrong order (arr[j] > arr[j + 1]).
     + If a swap is made, swapped is set to true.
   * **Optimization**:
     + If no swaps are made during a pass (swapped remains false), the array is already sorted, and the loop breaks early, improving efficiency.

**Problem Statement**

You are tasked with sorting a list of flight schedules based on their departure times. Each flight schedule is represented by the departure time in minutes and the destination city.

Write a program to take the flight schedules as input and sort them in ascending order of departure times using the selection sort algorithm.

**Input format :**

The first line of input consists of an integer **n,** denoting the number of flight schedules.

The following **n** lines of input consist of an integer **D**, representing the departure time, and a string **S,** representing the destination city. The departure time and destination city are separated by space.

**Output format :**

The output displays the sorted flight schedules in ascending order of departure times.

Each line should contain the departure time and the corresponding destination city, separated by a pipe (|) symbol.

**Refer to the sample output for the formatting specifications.**

**Code constraints :**

1 ≤ n ≤ 15

**Sample test cases :**

**Input 1 :**

4

1730 New York

1615 Los Angeles

1530 Chicago

1720 Miami

**Output 1 :**

Departure Time: 1530 minutes | Destination: Chicago

Departure Time: 1615 minutes | Destination: Los Angeles

Departure Time: 1720 minutes | Destination: Miami

Departure Time: 1730 minutes | Destination: New York

**Input 2 :**

3

1830 Tokyo

2359 London

1505 Paris

**Output 2 :**

Departure Time: 1505 minutes | Destination: Paris

Departure Time: 1830 minutes | Destination: Tokyo

Departure Time: 2359 minutes | Destination: London

**Solution:**

import java.util.Scanner;

class FlightSchedule {

    int departureTime; // Departure time in minutes

    String destination; // Destination of the flight

    // Constructor to initialize the departure time and destination

    FlightSchedule(int departureTime, String destination) {

        this.departureTime = departureTime;

        this.destination = destination;

    }

}

class FlightScheduler {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in); // Create a Scanner object to read input

        int n = scanner.nextInt(); // Read the number of flight schedules

        scanner.nextLine(); // Consume the newline character left by nextInt()

        FlightSchedule[] schedules = new FlightSchedule[n]; // Create an array to hold the flight schedules

        // Read each flight schedule and store it in the array

        for (int i = 0; i < n; i++) {

            int departureTime = scanner.nextInt(); // Read the departure time

            String destination = scanner.nextLine().trim(); // Read the destination and trim any leading/trailing spaces

            schedules[i] = new FlightSchedule(departureTime, destination); // Create a new FlightSchedule object and store it in the array

        }

        selectionSort(schedules); // Sort the flight schedules using selection sort

        // Print the sorted flight schedules

        for (FlightSchedule schedule : schedules) {

            System.out.println("Departure Time: " + schedule.departureTime + " minutes | Destination: " + schedule.destination);

        }

        scanner.close(); // Close the scanner

    }

    // Method to sort an array of FlightSchedule objects using selection sort

    public static void selectionSort(FlightSchedule[] schedules) {

        int n = schedules.length; // Get the length of the array

        // Iterate over the array to find the minimum element and swap it with the current element

        for (int i = 0; i < n - 1; i++) {

            int minIndex = i; // Assume the minimum element is at the current position

            for (int j = i + 1; j < n; j++) {

                // Find the index of the minimum element in the unsorted part of the array

                if (schedules[j].departureTime < schedules[minIndex].departureTime) {

                    minIndex = j;

                }

            }

            // Swap the minimum element found with the current element

            FlightSchedule temp = schedules[minIndex];

            schedules[minIndex] = schedules[i];

            schedules[i] = temp;

        }

    }

}

**Explanation**

1. **FlightSchedule Class**:
   * This class represents a flight schedule with two fields: departureTime (in minutes) and destination.
   * The constructor initializes these fields.
2. **FlightScheduler Class**:
   * This class contains the main method and a method to sort flight schedules.
3. **main Method**:
   * The main method is the entry point of the program.
   * Scanner scanner = new Scanner(System.in); creates a Scanner object for input.
   * int n = scanner.nextInt(); reads the number of flight schedules.
   * scanner.nextLine(); consumes the newline character left by nextInt().
   * FlightSchedule[] schedules = new FlightSchedule[n]; creates an array to hold the flight schedules.
   * A for loop reads each flight schedule's departure time and destination, creates a FlightSchedule object, and stores it in the array.
   * selectionSort(schedules); sorts the flight schedules using the selection sort algorithm.
   * A for loop prints the sorted flight schedules.
   * scanner.close(); closes the scanner.
4. **selectionSort Method**:
   * This method sorts an array of FlightSchedule objects using the selection sort algorithm.
   * int n = schedules.length; gets the length of the array.
   * The outer loop iterates over the array to find the minimum element in the unsorted part and swap it with the current element.
   * The inner loop finds the index of the minimum element in the unsorted part of the array.
   * The minimum element found is swapped with the current element.

Yes, you can simplify the solution by using Java's built-in sorting methods. The Arrays.sort method can be used to sort the array of FlightSchedule objects. To use this method, you'll need to make FlightSchedule implement the Comparable interface or use a Comparator. Here's how you can do it:

Solution using Comparable

First, make the FlightSchedule class implement the Comparable interface:

import java.util.Arrays;

import java.util.Scanner;

class FlightSchedule implements Comparable<FlightSchedule> {

    int departureTime;

    String destination;

    FlightSchedule(int departureTime, String destination) {

        this.departureTime = departureTime;

        this.destination = destination;

    }

    @Override

    public int compareTo(FlightSchedule other) {

        return Integer.compare(this.departureTime, other.departureTime);

    }

}

class FlightScheduler {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int n = scanner.nextInt();

        scanner.nextLine();

        FlightSchedule[] schedules = new FlightSchedule[n];

        for (int i = 0; i < n; i++) {

            int departureTime = scanner.nextInt();

            String destination = scanner.nextLine().trim();

            schedules[i] = new FlightSchedule(departureTime, destination);

        }

        Arrays.sort(schedules); // Sort using Arrays.sort which relies on the compareTo method

        for (FlightSchedule schedule : schedules) {

            System.out.println("Departure Time: " + schedule.departureTime + " minutes | Destination: " + schedule.destination);

        }

        scanner.close();

    }

}

Solution using Comparator

Alternatively, you can use a Comparator if you prefer not to modify the FlightSchedule class:

import java.util.Arrays;

import java.util.Comparator;

import java.util.Scanner;

class FlightSchedule {

    int departureTime;

    String destination;

    FlightSchedule(int departureTime, String destination) {

        this.departureTime = departureTime;

        this.destination = destination;

    }

}

class FlightScheduler {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int n = scanner.nextInt();

        scanner.nextLine();

        FlightSchedule[] schedules = new FlightSchedule[n];

        for (int i = 0; i < n; i++) {

            int departureTime = scanner.nextInt();

            String destination = scanner.nextLine().trim();

            schedules[i] = new FlightSchedule(departureTime, destination);

        }

        // Sort using Arrays.sort with a custom Comparator

        Arrays.sort(schedules, new Comparator<FlightSchedule>() {

            @Override

            public int compare(FlightSchedule fs1, FlightSchedule fs2) {

                return Integer.compare(fs1.departureTime, fs2.departureTime);

            }

        });

        for (FlightSchedule schedule : schedules) {

            System.out.println("Departure Time: " + schedule.departureTime + " minutes | Destination: " + schedule.destination);

        }

        scanner.close();

    }

}

Explanation

Using Comparable:

The FlightSchedule class implements the Comparable<FlightSchedule> interface.

The compareTo method is overridden to compare FlightSchedule objects based on their departureTime.

Arrays.sort(schedules) sorts the array using the natural order defined by the compareTo method.

Using Comparator:

A Comparator is provided to the Arrays.sort method to define the order based on departureTime.

This approach avoids modifying the FlightSchedule class.

Purpose

These approaches leverage Java's built-in sorting functionality, making the code simpler and more readable while ensuring the flight schedules are sorted by departure time in ascending order.

**Problem Statement**

You are given temperature data for a certain period. Your task is to sort the temperatures in ascending order using the **Bubble Sort** algorithm and determine the coldest and hottest days based on the sorted temperatures.

**Input format :**

The first line of input consists of an integer **n,** representing the number of days.

The second line of input consists of **n** space-separated float values, representing the temperatures for each day.

**Output format :**

The first line displays the float values representing the temperatures sorted in ascending order, separated by space, and rounded off to two decimal places.

The second line displays the "Coldest day: X degrees" where X is a float value, representing the coldest day temperature and rounded off to two decimal places.

The third line displays the "Hottest day: Y degrees" where Y is a float value, representing the hottest day temperature and rounded off to two decimal places.

**Refer to the sample output for the exact format.**

**Code constraints :**

1 ≤ n ≤ 10

1.0 ≤ temperatures ≤ 100.0

**Sample test cases :**

**Input 1 :**

7

30.61 24.22 31.54 18.3 30.63 21.14 32.42

**Output 1 :**

18.30 21.14 24.22 30.61 30.63 31.54 32.42

Coldest day: 18.30 degrees

Hottest day: 32.42 degrees

**Input 2 :**

5

21.1 21.9 21.13 19.94 21.15

**Output 2 :**

19.94 21.10 21.13 21.15 21.90

Coldest day: 19.94 degrees

Hottest day: 21.90 degrees

**Solution:**

import java.util.Scanner;

import java.text.DecimalFormat;

class TemperatureSorter {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in); // Create a Scanner object to read input

        int n = scanner.nextInt(); // Read the number of temperatures

        float[] temperatures = new float[n]; // Create an array to hold the temperatures

        // Read each temperature and store it in the array

        for (int i = 0; i < n; i++) {

            temperatures[i] = scanner.nextFloat();

        }

        bubbleSort(temperatures); // Sort the temperatures using bubble sort

        DecimalFormat df = new DecimalFormat("#.00"); // Create a DecimalFormat object to format the temperatures

        // Print the sorted temperatures

        for (int i = 0; i < n; i++) {

            System.out.print(df.format(temperatures[i]) + " ");

        }

        System.out.println(); // Print a newline

        // Print the coldest day temperature

        System.out.println("Coldest day: " + df.format(temperatures[0]) + " degrees");

        // Print the hottest day temperature

        System.out.println("Hottest day: " + df.format(temperatures[n - 1]) + " degrees");

        scanner.close(); // Close the scanner

    }

    // Method to sort an array of floats using bubble sort

    public static void bubbleSort(float[] arr) {

        int n = arr.length; // Get the length of the array

        boolean swapped; // Variable to check if any swaps were made in a pass

        // Outer loop to control the number of passes

        for (int i = 0; i < n - 1; i++) {

            swapped = false; // Reset swapped to false for each pass

            // Inner loop to compare adjacent elements and swap if needed

            for (int j = 0; j < n - 1 - i; j++) {

                if (arr[j] > arr[j + 1]) { // If the current element is greater than the next element

                    // Swap arr[j] and arr[j + 1]

                    float temp = arr[j];

                    arr[j] = arr[j + 1];

                    arr[j + 1] = temp;

                    swapped = true; // Set swapped to true if a swap is made

                }

            }

            // If no swaps were made in this pass, the array is already sorted

            if (!swapped) break;

        }

    }

}

**Explanation**

1. **Imports and Class Definition**:
   * java.util.Scanner is imported to read input from the user.
   * java.text.DecimalFormat is imported to format the temperatures to two decimal places.
2. **main Method**:
   * The main method is the entry point of the program.
   * Scanner scanner = new Scanner(System.in); creates a Scanner object for input.
   * int n = scanner.nextInt(); reads the number of temperatures.
   * float[] temperatures = new float[n]; creates an array to hold the temperatures.
   * A for loop reads each temperature and stores it in the array.
   * bubbleSort(temperatures); sorts the temperatures using the bubble sort algorithm.
   * DecimalFormat df = new DecimalFormat("#.00"); creates a DecimalFormat object to format the temperatures.
   * A for loop prints the sorted temperatures formatted to two decimal places.
   * System.out.println("Coldest day: " + df.format(temperatures[0]) + " degrees"); prints the coldest day temperature.
   * System.out.println("Hottest day: " + df.format(temperatures[n - 1]) + " degrees"); prints the hottest day temperature.
   * scanner.close(); closes the scanner.
3. **bubbleSort Method**:
   * This method sorts an array of floats using the bubble sort algorithm.
   * int n = arr.length; gets the length of the array.
   * The outer loop runs n - 1 times, controlling the number of passes.
   * boolean swapped; is a flag to check if any swaps were made during a pass.
   * The inner loop compares adjacent elements and swaps them if they are in the wrong order (arr[j] > arr[j + 1]).
   * If a swap is made, swapped is set to true.
   * If no swaps are made during a pass (swapped remains false), the array is already sorted, and the loop breaks early, improving efficiency.
4. import java.util.Arrays;
5. import java.util.Scanner;
6. import java.text.DecimalFormat;
7. class TemperatureSorter {
8. public static void main(String[] args) {
9. Scanner scanner = new Scanner(System.in);
10. int n = scanner.nextInt();
11. float[] temperatures = new float[n];
12. for (int i = 0; i < n; i++) {
13. temperatures[i] = scanner.nextFloat();
14. }
15. Arrays.sort(temperatures); // Sort using built-in Arrays.sort
16. DecimalFormat df = new DecimalFormat("#.00");
17. for (int i = 0; i < n; i++) {
18. System.out.print(df.format(temperatures[i]) + " ");
19. }
20. System.out.println();
21. System.out.println("Coldest day: " + df.format(temperatures[0]) + " degrees");
22. System.out.println("Hottest day: " + df.format(temperatures[n - 1]) + " degrees");
23. scanner.close();
24. }
25. }

**Explanation**

1. **Using Arrays.sort**:
   * Arrays.sort(temperatures); sorts the array of temperatures in ascending order.
   * This approach eliminates the need to implement the bubble sort algorithm manually, simplifying the code and improving efficiency.

import java.util.Arrays;

import java.util.Scanner;

class TemperatureSorter {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int n = scanner.nextInt();

        float[] temperatures = new float[n];

        for (int i = 0; i < n; i++) {

            temperatures[i] = scanner.nextFloat();

        }

        Arrays.sort(temperatures); // Sort the temperatures

        for (float temp : temperatures) {

            // Print each temperature formatted to two decimal places

            System.out.printf("%.2f ", temp);

        }

        System.out.println();

        // Print the coldest and hottest day temperatures

        System.out.printf("Coldest day: %.2f degrees\n", temperatures[0]);

        System.out.printf("Hottest day: %.2f degrees\n", temperatures[n - 1]);

        scanner.close();

    }

}

**Printf using**

import java.util.Arrays;

import java.util.Scanner;

class TemperatureSorter {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in); // Create a Scanner object to read input

        int n = scanner.nextInt(); // Read the number of temperatures

        float[] temperatures = new float[n]; // Create an array to hold the temperatures

        // Read each temperature and store it in the array

        for (int i = 0; i < n; i++) {

            temperatures[i] = scanner.nextFloat();

        }

        bubbleSort(temperatures); // Sort the temperatures using bubble sort

        // Print the sorted temperatures

        for (int i = 0; i < n; i++) {

            System.out.printf("%.2f ", temperatures[i]); // Format each temperature to two decimal places

        }

        System.out.println(); // Print a newline

        // Print the coldest day temperature

        System.out.printf("Coldest day: %.2f degrees\n", temperatures[0]);

        // Print the hottest day temperature

        System.out.printf("Hottest day: %.2f degrees\n", temperatures[n - 1]);

        scanner.close(); // Close the scanner

    }

    // Method to sort an array of floats using bubble sort

    public static void bubbleSort(float[] arr) {

        int n = arr.length; // Get the length of the array

        boolean swapped; // Variable to check if any swaps were made in a pass

        // Outer loop to control the number of passes

        for (int i = 0; i < n - 1; i++) {

            swapped = false; // Reset swapped to false for each pass

            // Inner loop to compare adjacent elements and swap if needed

            for (int j = 0; j < n - 1 - i; j++) {

                if (arr[j] > arr[j + 1]) { // If the current element is greater than the next element

                    // Swap arr[j] and arr[j + 1]

                    float temp = arr[j];

                    arr[j] = arr[j + 1];

                    arr[j + 1] = temp;

                    swapped = true; // Set swapped to true if a swap is made

                }

            }

            // If no swaps were made in this pass, the array is already sorted

            if (!swapped) break;

        }

    }

}

import java.util.Arrays;

import java.util.Scanner;

class TemperatureSorter {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in); // Create a Scanner object to read input

        int n = scanner.nextInt(); // Read the number of temperatures

        float[] temperatures = new float[n]; // Create an array to hold the temperatures

        // Read each temperature and store it in the array

        for (int i = 0; i < n; i++) {

            temperatures[i] = scanner.nextFloat();

        }

        bubbleSort(temperatures); // Sort the temperatures using bubble sort

        // Print the sorted temperatures

        for (int i = 0; i < n; i++) {

            System.out.print(String.format("%.2f", temperatures[i]) + " ");

        }

        System.out.println(); // Print a newline

        // Print the coldest day temperature

        System.out.println("Coldest day: " + String.format("%.2f", temperatures[0]) + " degrees");

        // Print the hottest day temperature

        System.out.println("Hottest day: " + String.format("%.2f", temperatures[n - 1]) + " degrees");

        scanner.close(); // Close the scanner

    }

    // Method to sort an array of floats using bubble sort

    public static void bubbleSort(float[] arr) {

        int n = arr.length; // Get the length of the array

        boolean swapped; // Variable to check if any swaps were made in a pass

        // Outer loop to control the number of passes

        for (int i = 0; i < n - 1; i++) {

            swapped = false; // Reset swapped to false for each pass

            // Inner loop to compare adjacent elements and swap if needed

            for (int j = 0; j < n - 1 - i; j++) {

                if (arr[j] > arr[j + 1]) { // If the current element is greater than the next element

                    // Swap arr[j] and arr[j + 1]

                    float temp = arr[j];

                    arr[j] = arr[j + 1];

                    arr[j + 1] = temp;

                    swapped = true; // Set swapped to true if a swap is made

                }

            }

            // If no swaps were made in this pass, the array is already sorted

            if (!swapped) break;

        }

    }

}

**Using String.format:**

* **String.format("%.2f", temperatures[i]) formats each temperature to two decimal places as a string.**

**Benefits**

* **Simplicity**: String.format is straightforward for simple formatting needs and avoids the overhead of creating and managing a DecimalFormat object.
* **Built-in**: String.format is part of the Java standard library and does not require importing additional packages for basic formatting.

By using String.format or System.out.printf, you can efficiently format floating-point numbers to a specified number of decimal places without needing the DecimalFormat class.

**Explanation of Changes**

1. **Printing Sorted Temperatures**:
   * Replaced System.out.print(String.format("%.2f", temperatures[i]) + " "); with System.out.printf("%.2f ", temperatures[i]);.
2. **Printing Coldest and Hottest Temperatures**:
   * Replaced System.out.println("Coldest day: " + String.format("%.2f", temperatures[0]) + " degrees"); with System.out.printf("Coldest day: %.2f degrees%n", temperatures[0]);.
   * Replaced System.out.println("Hottest day: " + String.format("%.2f", temperatures[n - 1]) + " degrees"); with System.out.printf("Hottest day: %.2f degrees%n", temperatures[n - 1]);.

**Summary**

Using System.out.printf provides a direct way to format and print the data, which simplifies the code by eliminating the need for String.format. This approach is especially handy for straightforward formatting tasks and helps keep the code concise.