LAPORAN PRAKTIKUM PEMROGRAMAN LANJUT

Nama : Akhmad Syafiul Anam
NIM : 245150707111012
Kelas : TI - A

Bab : BAB 11 (Sorting dan Searching)
Asisten : 1. Ketut Bagus Wedanta Ananda Murti
2. Gantang Satria Yudha.

LANGKAH 1

A. Soal

Tugas Praktikum

Apabila diketahui data anggota tim futsal sebagai berikut:

No	Tim A		Tim B	
	Tinggi Badan (cm)	Berat Badan (kg)	Tinggi Badan (cm)	Berat Badan (kg)
1	168	50	170	66
2	170	60	167	60
3	165	56	165	59
4	168	55	166	58
5	172	60	168	58
6	170	70	175	71
7	169	66	172	68
8	165	56	171	68
9	171	72	168	65
10	166	56	169	60

1. Dengan program java, urutkan data pemain diantara kedua tim tersebut: a. Berdasarkan Tinggi Badannya secara Ascending/menaik dan

Descending/menurun

2. Berdasarkan Berat Badannya secara Ascending/menaik dan Descending/menurun
3. Cari nilai maksimum dan minimum Tinggi Badan dan Berat Badan untuk pemain dari masing-masing tim.
4. Copy seluruh anggota Tim B ke Tim C yang baru dibentuk

2. Buatlah implementasi Binary Search dalam program java berdasarkan kondisi berikut:

- Implementasikan ArrayList untuk menyimpan data tim A dan tim B dalam bentuk ArrayList terpisah.
- Dari data tim B, dicari jumlah pemain yang mempunyai tinggi badan 168 cm dan 160 cm.
- Dari data tim A, dicari jumlah pemain yang mempunyai berat badan 56 kg dan 53 kg.
- Ingin diketahui apakah pemain di Tim A ada yang mempunyai tinggi badan atau berat badan yang sama dengan pemain di Tim B?

B. Screenshoot

Player.Java	HeightComparator.java

```java
// Player.java
// Class untuk merepresentasikan data pemain futsal

public class Player {
    private int height; // tinggi badan (cm)
    private int weight; // berat badan (kg)
    private String team; // nama tim
    private int playerNumber; // nomor pemain

    // Constructor
    public Player(int playerNumber, int height, int weight, String team) {
        this.playerNumber = playerNumber;
        this.height = height;
        this.weight = weight;
        this.team = team;
    }

    // Getter methods
    public int getHeight() {
        return height;
    }

    public int getWeight() {
        return weight;
    }

    public String getTeam() {
        return team;
    }

    public int getPlayerNumber() {
        return playerNumber;
    }

    // Setter methods
    public void setHeight(int height) {
        this.height = height;
    }

    public void setWeight(int weight) {
        this.weight = weight;
    }

    public void setTeam(String team) {
        this.team = team;
    }

    public void setPlayerNumber(int playerNumber) {
        this.playerNumber = playerNumber;
    }

    // Override toString untuk display yang mudah dibaca
    @Override
    public String toString() {
        return String.format("Player %d (%s): Height=%dcm, Weight=%dkg",
                             playerNumber, team, height, weight);
    }

    // Method untuk membuat copy player dengan tim berbeda
    public Player copyToTeam(String newTeam) {
        return new Player(this.playerNumber, this.height, this.weight, newTeam);
    }

    // Method untuk membandingkan kesamaan data (tanpa mempertimbangkan tim dan nomor)
    public boolean hasSamePhysicalData(Player other) {
        return this.height == other.height && this.weight == other.weight;
    }
}
```

```java
// HeightComparator.java
// Comparator untuk sorting berdasarkan tinggi badan

import java.util.Comparator;

public class HeightComparator implements Comparator<Player> {
    private boolean ascending;

    // Constructor dengan parameter ascending/descending
    public HeightComparator(boolean ascending) {
        this.ascending = ascending;
    }

    // Default constructor (ascending)
    public HeightComparator() {
        this.ascending = true;
    }

    @Override
    public int compare(Player p1, Player p2) {
        if (ascending) {
            return Integer.compare(p1.getHeight(), p2.getHeight());
        } else {
            return Integer.compare(p2.getHeight(), p1.getHeight());
        }
    }

    // Method untuk mengubah urutan sorting
    public void setAscending(boolean ascending) {
        this.ascending = ascending;
    }

    public boolean isAscending() {
        return ascending;
    }
}
```

WeightComparator.java	TeamDataManager.java

```java
// WeightComparator.java
// Comparator untuk sorting berdasarkan berat badan

import java.util.Comparator;

public class WeightComparator implements Comparator<Player> {
    private boolean ascending;

    // Constructor dengan parameter ascending/descending
    public WeightComparator(boolean ascending) {
        this.ascending = ascending;
    }

    // Default constructor (ascending)
    public WeightComparator() {
        this.ascending = true;
    }

    @Override
    public int compare(Player p1, Player p2) {
        if (ascending) {
            return Integer.compare(p1.getWeight(), p2.getWeight());
        } else {
            return Integer.compare(p2.getWeight(), p1.getWeight());
        }
    }

    // Method untuk mengubah urutan sorting
    public void setAscending(boolean ascending) {
        this.ascending = ascending;
    }

    public boolean isAscending() {
        return ascending;
    }
}
```

```java
// TeamDataManager.java
// Class untuk mengelola data tim dan operasi dasar

import java.util.ArrayList;

public class TeamDataManager {
    private ArrayList<Player> teamA;
    private ArrayList<Player> teamB;
    private ArrayList<Player> teamC;

    public TeamDataManager() {
        initializeTeams();
    }

    // Inisialisasi data tim berdasarkan data yang diberikan
    private void initializeTeams() {
        teamA = new ArrayList<>();
        teamB = new ArrayList<>();
        teamC = new ArrayList<>();

        // Data Tim A: {tinggi, berat}
        int[][] dataTimA = {
            {168, 50}, {170, 60}, {165, 56}, {168, 55}, {172, 60},
            {170, 70}, {169, 66}, {165, 56}, {171, 72}, {166, 56}
        };

        // Data Tim B: {tinggi, berat}
        int[][] dataTimB = {
            {170, 66}, {167, 60}, {165, 59}, {166, 58}, {168, 58},
            {175, 71}, {172, 68}, {171, 68}, {168, 65}, {169, 60}
        };

        // Populate Tim A
        for (int i = 0; i < dataTimA.length; i++) {
            teamA.add(new Player(i + 1, dataTimA[i][0], dataTimA[i][1], "Tim A"));
        }

        // Populate Tim B
        for (int i = 0; i < dataTimB.length; i++) {
            teamB.add(new Player(i + 1, dataTimB[i][0], dataTimB[i][1], "Tim B"));
        }
    }

    // Getter methods untuk mengakses data tim
    public ArrayList<Player> getTeamA() {
        return new ArrayList<>(teamA); // Return copy untuk keamanan
    }

    public ArrayList<Player> getTeamB() {
        return new ArrayList<>(teamB); // Return copy untuk keamanan
    }

    public ArrayList<Player> getTeamC() {
        return new ArrayList<>(teamC); // Return copy untuk keamanan
    }

    // Method untuk copy Tim B ke Tim C
    public void copyTeamBToTeamC() {
        teamC.clear(); // Clear existing data
        for (Player player : teamB) {
            teamC.add(player.copyToTeam("Tim C"));
        }
    }

    // Method untuk menampilkan data tim
    public void displayTeam(String teamName) {
        ArrayList<Player> team;

        switch (teamName.toUpperCase()) {
            case "TIM A":
            case "A":
                team = teamA;
                break;
            case "TIM B":
            case "B":
                team = teamB;
                break;
            case "TIM C":
            case "C":
                team = teamC;
                break;
            default:
                System.out.println("Tim tidak ditemukan: " + teamName);
                return;
        }

        System.out.println("\n--- " + teamName + " ---");
        if (team.isEmpty()) {
            System.out.println("Tim kosong");
        } else {
            for (Player p : team) {
                System.out.println(p);
            }
        }
    }

    // Method untuk menampilkan semua tim
    public void displayAllTeams() {
        System.out.println("=== DATA TIM FUTSAL ===");
        displayTeam("Tim A");
        displayTeam("Tim B");
        if (!teamC.isEmpty()) {
            displayTeam("Tim C");
        }
    }

    // Method untuk mendapatkan gabungan semua pemain
    public ArrayList<Player> getAllPlayers() {
        ArrayList<Player> allPlayers = new ArrayList<>();
        allPlayers.addAll(teamA);
        allPlayers.addAll(teamB);
        if (!teamC.isEmpty()) {
            allPlayers.addAll(teamC);
        }
        return allPlayers;
    }

    // Method untuk mendapatkan jumlah pemain per tim
    public void displayTeamSizes() {
        System.out.println("\n=== JUMLAH PEMAIN ===");
        System.out.println("Tim A: " + teamA.size() + " pemain");
        System.out.println("Tim B: " + teamB.size() + " pemain");
        System.out.println("Tim C: " + teamC.size() + " pemain");
    }
}
```

SortingOperations.java

```java
// SortingOperations.java
// Class untuk menangani operasi sorting dan pencarian min/max

import java.util.ArrayList;
import java.util.Collections;

public class SortingOperations {
    private TeamDataManager dataManager;

    public SortingOperations(TeamDataManager dataManager) {
        this.dataManager = dataManager;
    }

    // 1a. Sorting berdasarkan tinggi badan
    public void sortByHeight() {
        System.out.println("\n=== SORTING BERDASARKAN TINGGI BADAN ===");

        // Gabungkan kedua tim
        ArrayList<Player> allPlayers = new ArrayList<>();
        allPlayers.addAll(dataManager.getTeamA());
        allPlayers.addAll(dataManager.getTeamB());

        // Ascending
        Collections.sort(allPlayers, new HeightComparator(true));
        System.out.println("\n--- Tinggi Badan (Ascending) ---");
        displayPlayerList(allPlayers);

        // Descending
        Collections.sort(allPlayers, new HeightComparator(false));
        System.out.println("\n--- Tinggi Badan (Descending) ---");
        displayPlayerList(allPlayers);
    }

    // 1b. Sorting berdasarkan berat badan
    public void sortByWeight() {
        System.out.println("\n=== SORTING BERDASARKAN BERAT BADAN ===");

        // Gabungkan kedua tim
        ArrayList<Player> allPlayers = new ArrayList<>();
        allPlayers.addAll(dataManager.getTeamA());
        allPlayers.addAll(dataManager.getTeamB());

        // Ascending
        Collections.sort(allPlayers, new WeightComparator(true));
        System.out.println("\n--- Berat Badan (Ascending) ---");
        displayPlayerList(allPlayers);

        // Descending
        Collections.sort(allPlayers, new WeightComparator(false));
        System.out.println("\n--- Berat Badan (Descending) ---");
        displayPlayerList(allPlayers);
    }

    // 1c. Cari nilai max dan min tinggi badan dan berat badan
    public void findMinMaxValues() {
        System.out.println("\n=== NILAI MAKSIMUM DAN MINIMUM ===");

        ArrayList<Player> teamA = dataManager.getTeamA();
        ArrayList<Player> teamB = dataManager.getTeamB();

        // Tim A
        System.out.println("\n--- Tim A ---");
        findMinMaxForTeam(teamA, "Tim A");

        // Tim B
        System.out.println("\n--- Tim B ---");
        findMinMaxForTeam(teamB, "Tim B");

        // Gabungan kedua tim
        ArrayList<Player> allPlayers = new ArrayList<>();
        allPlayers.addAll(teamA);
        allPlayers.addAll(teamB);
        System.out.println("\n--- Gabungan Tim A & B ---");
        findMinMaxForTeam(allPlayers, "Gabungan");
    }

    // Helper method untuk mencari min/max satu tim
    private void findMinMaxForTeam(ArrayList<Player> team, String teamName) {
        if (team.isEmpty()) {
            System.out.println(teamName + " kosong");
            return;
        }

        // Cari min/max tinggi badan
        Player minHeightPlayer = Collections.min(team, new HeightComparator(true));
        Player maxHeightPlayer = Collections.max(team, new HeightComparator(true));

        // Cari min/max berat badan
        Player minWeightPlayer = Collections.min(team, new WeightComparator(true));
        Player maxWeightPlayer = Collections.max(team, new WeightComparator(true));

        System.out.println("Tinggi Badan:");
        System.out.println("  Min: " + minHeightPlayer.getHeight() + "cm (Player " +
                           minHeightPlayer.getPlayerNumber() + " - " + minHeightPlayer.getTeam() + ")");
        System.out.println("  Max: " + maxHeightPlayer.getHeight() + "cm (Player " +
                           maxHeightPlayer.getPlayerNumber() + " - " + maxHeightPlayer.getTeam() + ")");

        System.out.println("Berat Badan:");
        System.out.println("  Min: " + minWeightPlayer.getWeight() + "kg (Player " +
                           minWeightPlayer.getPlayerNumber() + " - " + minWeightPlayer.getTeam() + ")");
        System.out.println("  Max: " + maxWeightPlayer.getWeight() + "kg (Player " +
                           maxWeightPlayer.getPlayerNumber() + " - " + maxWeightPlayer.getTeam() + ")");
    }

    // Helper method untuk menampilkan daftar pemain
    private void displayPlayerList(ArrayList<Player> players) {
        for (Player p : players) {
            System.out.println(p);
        }
    }

    // Method tambahan: sorting hanya satu tim
    public void sortSingleTeam(String teamName, String criteria, boolean ascending) {
        ArrayList<Player> team;

        switch (teamName.toUpperCase()) {
            case "TIM A":
            case "A":
                team = dataManager.getTeamA();
                break;
            case "TIM B":
            case "B":
                team = dataManager.getTeamB();
                break;
            case "TIM C":
            case "C":
                team = dataManager.getTeamC();
                break;
            default:
                System.out.println("Tim tidak ditemukan: " + teamName);
                return;
        }

        System.out.println("\n--- Sorting " + teamName + " berdasarkan " + criteria +
                           (ascending ? " (Ascending)" : " (Descending)") + " ---");

        if (criteria.equalsIgnoreCase("height") || criteria.equalsIgnoreCase("tinggi")) {
            Collections.sort(team, new HeightComparator(ascending));
        } else if (criteria.equalsIgnoreCase("weight") || criteria.equalsIgnoreCase("berat")) {
            Collections.sort(team, new WeightComparator(ascending));
        } else {
            System.out.println("Kriteria tidak valid. Gunakan 'height' atau 'weight'");
            return;
        }

        displayPlayerList(team);
    }
}
```

Player.java

```java
// Player.java
// Class untuk merepresentasikan data pemain futsal

public class Player {
    private int height; // tinggi badan (cm)
    private int weight; // berat badan (kg)
    private String team; // nama tim
    private int playerNumber; // nomor pemain

    // Constructor
    public Player(int playerNumber, int height, int weight, String team) {
        this.playerNumber = playerNumber;
        this.height = height;
        this.weight = weight;
        this.team = team;
    }

    // Getter methods
    public int getHeight() {
        return height;
    }

    public int getWeight() {
        return weight;
    }

    public String getTeam() {
        return team;
    }

    public int getPlayerNumber() {
        return playerNumber;
    }

    // Setter methods
    public void setHeight(int height) {
        this.height = height;
    }

    public void setWeight(int weight) {
        this.weight = weight;
    }

    public void setTeam(String team) {
        this.team = team;
    }

    public void setPlayerNumber(int playerNumber) {
        this.playerNumber = playerNumber;
    }

    // Override toString untuk display yang mudah dibaca
    @Override
    public String toString() {
        return String.format("Player %d (%s): Height=%dcm, Weight=%dkg",
                             playerNumber, team, height, weight);
    }

    // Method untuk membuat copy player dengan tim berbeda
    public Player copyToTeam(String newTeam) {
        return new Player(this.playerNumber, this.height, this.weight, newTeam);
    }

    // Method untuk membandingkan kesamaan data (tanpa mempertimbangkan tim dan nomor)
    public boolean hasSamePhysicalData(Player other) {
        return this.height == other.height && this.weight == other.weight;
    }
}
```

SearchOperation.java

C. Syntax

```java
// Player.java
// Class untuk merepresentasikan data pemain futsal

public class Player {
    private int height; // tinggi badan (cm)
    private int weight; // berat badan (kg)
    private String team; // nama tim
    private int playerNumber; // nomor pemain

    // Constructor
    public Player(int playerNumber, int height, int weight, String team) {
        this.playerNumber = playerNumber;
        this.height = height;
        this.weight = weight;
        this.team = team;
    }

    // Getter methods
    public int getHeight() {
        return height;
    }

    public int getWeight() {
        return weight;
    }

    public String getTeam() {
        return team;
    }

    public int getPlayerNumber() {
        return playerNumber;
    }

    // Setter methods
    public void setHeight(int height) {
        this.height = height;
    }
```

```
39.
40.     public void setWeight(int weight) {
41.         this.weight = weight;
42.     }
43.
44.     public void setTeam(String team) {
45.         this.team = team;
46.     }
47.
48.     public void setPlayerNumber(int playerNumber) {
49.         this.playerNumber = playerNumber;
50.     }
51.
52.     // Override toString untuk display yang mudah dibaca
53.     @Override
54.     public String toString() {
55.         return String.format("Player %d (%s):
    Height=%dcm, Weight=%dkg",
56.                              playerNumber, team, height,
    weight);
57.     }
58.
59.     // Method untuk membuat copy player dengan tim
    berbeda
60.     public Player copyToTeam(String newTeam) {
61.         return new Player(this.playerNumber,
    this.height, this.weight, newTeam);
62.     }
63.
64.     // Method untuk membandingkan kesamaan data (tanpa
    mempertimbangkan tim dan nomor)
65.     public boolean hasSamePhysicalData(Player other) {
66.         return this.height == other.height &&
    this.weight == other.weight;
67.     }
68.}
```

```
1. // HeightComparator.java
2. // Comparator untuk sorting berdasarkan tinggi badan
3.
4. import java.util.Comparator;
5.
```

```java
public class HeightComparator implements Comparator<Player> {
    private boolean ascending;

    // Constructor dengan parameter ascending/descending
    public HeightComparator(boolean ascending) {
        this.ascending = ascending;
    }

    // Default constructor (ascending)
    public HeightComparator() {
        this.ascending = true;
    }

    @Override
    public int compare(Player p1, Player p2) {
        if (ascending) {
            return Integer.compare(p1.getHeight(), p2.getHeight());
        } else {
            return Integer.compare(p2.getHeight(), p1.getHeight());
        }
    }

    // Method untuk mengubah urutan sorting
    public void setAscending(boolean ascending) {
        this.ascending = ascending;
    }

    public boolean isAscending() {
        return ascending;
    }
}
```

```java
// WeightComparator.java
// Comparator untuk sorting berdasarkan berat badan

import java.util.Comparator;

public class WeightComparator implements Comparator<Player> {
```

```
    private boolean ascending;

    // Constructor dengan parameter ascending/descending
    public WeightComparator(boolean ascending) {
        this.ascending = ascending;
    }

    // Default constructor (ascending)
    public WeightComparator() {
        this.ascending = true;
    }

    @Override
    public int compare(Player p1, Player p2) {
        if (ascending) {
            return Integer.compare(p1.getWeight(),
p2.getWeight());
        } else {
            return Integer.compare(p2.getWeight(),
p1.getWeight());
        }
    }

    // Method untuk mengubah urutan sorting
    public void setAscending(boolean ascending) {
        this.ascending = ascending;
    }

    public boolean isAscending() {
        return ascending;
    }
}
```

```
// TeamDataManager.java
// Class untuk mengelola data tim dan operasi dasar

import java.util.ArrayList;

public class TeamDataManager {
    private ArrayList<Player> teamA;
    private ArrayList<Player> teamB;
    private ArrayList<Player> teamC;
```

```
10.
11.     public TeamDataManager() {
12.         initializeTeams();
13.     }
14.
15.     // Inisialisasi data tim berdasarkan data yang
    diberikan
16.     private void initializeTeams() {
17.         teamA = new ArrayList<>();
18.         teamB = new ArrayList<>();
19.         teamC = new ArrayList<>();
20.
21.         // Data Tim A: {tinggi, berat}
22.         int[][] dataTimA = {
23.             {168, 50}, {170, 60}, {165, 56}, {168, 55},
    {172, 60},
24.             {170, 70}, {169, 66}, {165, 56}, {171, 72},
    {166, 56}
25.         };
26.
27.         // Data Tim B: {tinggi, berat}
28.         int[][] dataTimB = {
29.             {170, 66}, {167, 60}, {165, 59}, {166, 58},
    {168, 58},
30.             {175, 71}, {172, 68}, {171, 68}, {168, 65},
    {169, 60}
31.         };
32.
33.         // Populate Tim A
34.         for (int i = 0; i < dataTimA.length; i++) {
35.             teamA.add(new Player(i + 1, dataTimA[i][0],
    dataTimA[i][1], "Tim A"));
36.         }
37.
38.         // Populate Tim B
39.         for (int i = 0; i < dataTimB.length; i++) {
40.             teamB.add(new Player(i + 1, dataTimB[i][0],
    dataTimB[i][1], "Tim B"));
41.         }
42.     }
43.
44.     // Getter methods untuk mengakses data tim
45.     public ArrayList<Player> getTeamA() {
```

```
        return new ArrayList<>(teamA); // Return copy untuk keamanan
    }

    public ArrayList<Player> getTeamB() {
        return new ArrayList<>(teamB); // Return copy untuk keamanan
    }

    public ArrayList<Player> getTeamC() {
        return new ArrayList<>(teamC); // Return copy untuk keamanan
    }

    // Method untuk copy Tim B ke Tim C
    public void copyTeamBToTeamC() {
        teamC.clear(); // Clear existing data
        for (Player player : teamB) {
            teamC.add(player.copyToTeam("Tim C"));
        }
    }

    // Method untuk menampilkan data tim
    public void displayTeam(String teamName) {
        ArrayList<Player> team;

        switch (teamName.toUpperCase()) {
            case "TIM A":
            case "A":
                team = teamA;
                break;
            case "TIM B":
            case "B":
                team = teamB;
                break;
            case "TIM C":
            case "C":
                team = teamC;
                break;
            default:
                System.out.println("Tim tidak ditemukan: " + teamName);
                return;
```

```
        }

        System.out.println("\n--- " + teamName + "
    ---");
        if (team.isEmpty()) {
            System.out.println("Tim kosong");
        } else {
            for (Player p : team) {
                System.out.println(p);
            }
        }
    }

    // Method untuk menampilkan semua tim
    public void displayAllTeams() {
        System.out.println("=== DATA TIM FUTSAL ===");
           displayTeam("Tim A");
           displayTeam("Tim B");
           if (!teamC.isEmpty()) {
               displayTeam("Tim C");
           }
       }

       // Method untuk mendapatkan gabungan semua pemain
       public ArrayList<Player> getAllPlayers() {
           ArrayList<Player> allPlayers = new
    ArrayList<>();
           allPlayers.addAll(teamA);
           allPlayers.addAll(teamB);
           if (!teamC.isEmpty()) {
               allPlayers.addAll(teamC);
           }
           return allPlayers;
       }

       // Method untuk mendapatkan jumlah pemain per tim
       public void displayTeamSizes() {
           System.out.println("\n=== JUMLAH PEMAIN
    ===");
           System.out.println("Tim A: " + teamA.size() +
    " pemain");
           System.out.println("Tim B: " + teamB.size() +
    " pemain");
```

```
123.           System.out.println("Tim C: " + teamC.size() +
   " pemain");
124.       }
125.   }
```

```
1. // SortingOperations.java
2. // Class untuk menangani operasi sorting dan pencarian
   min/max
3.
4. import java.util.ArrayList;
5. import java.util.Collections;
6.
7. public class SortingOperations {
8.     private TeamDataManager dataManager;
9.
10.    public SortingOperations(TeamDataManager
   dataManager) {
11.        this.dataManager = dataManager;
12.    }
13.
14.    // 1a. Sorting berdasarkan tinggi badan
15.    public void sortByHeight() {
16.        System.out.println("\n=== SORTING BERDASARKAN
   TINGGI BADAN ===");
17.
18.        // Gabungkan kedua tim
19.        ArrayList<Player> allPlayers = new
   ArrayList<>();
20.        allPlayers.addAll(dataManager.getTeamA());
21.        allPlayers.addAll(dataManager.getTeamB());
22.
23.        // Ascending
24.        Collections.sort(allPlayers, new
   HeightComparator(true));
25.        System.out.println("\n--- Tinggi Badan
   (Ascending) ---");
26.        displayPlayerList(allPlayers);
27.
28.        // Descending
29.        Collections.sort(allPlayers, new
   HeightComparator(false));
30.        System.out.println("\n--- Tinggi Badan
```

```
(Descending) ---");
31.         displayPlayerList(allPlayers);
32.     }
33.
34.     // 1b. Sorting berdasarkan berat badan
35.     public void sortByWeight() {
36.         System.out.println("\n=== SORTING BERDASARKAN
BERAT BADAN ===");
37.
38.         // Gabungkan kedua tim
39.         ArrayList<Player> allPlayers = new
ArrayList<>();
40.         allPlayers.addAll(dataManager.getTeamA());
41.         allPlayers.addAll(dataManager.getTeamB());
42.
43.         // Ascending
44.         Collections.sort(allPlayers, new
WeightComparator(true));
45.         System.out.println("\n--- Berat Badan
(Ascending) ---");
46.         displayPlayerList(allPlayers);
47.
48.         // Descending
49.         Collections.sort(allPlayers, new
WeightComparator(false));
50.         System.out.println("\n--- Berat Badan
(Descending) ---");
51.         displayPlayerList(allPlayers);
52.     }
53.
54.     // 1c. Cari nilai max dan min tinggi badan dan berat
badan
55.     public void findMinMaxValues() {
56.         System.out.println("\n=== NILAI MAKSIMUM DAN
MINIMUM ===");
57.
58.         ArrayList<Player> teamA =
dataManager.getTeamA();
59.         ArrayList<Player> teamB =
dataManager.getTeamB();
60.
61.         // Tim A
62.         System.out.println("\n--- Tim A ---");
```

```
63.         findMinMaxForTeam(teamA, "Tim A");
64.
65.         // Tim B
66.         System.out.println("\n--- Tim B ---");
67.         findMinMaxForTeam(teamB, "Tim B");
68.
69.         // Gabungan kedua tim
70.         ArrayList<Player> allPlayers = new
    ArrayList<>();
71.         allPlayers.addAll(teamA);
72.         allPlayers.addAll(teamB);
73.         System.out.println("\n--- Gabungan Tim A & B
    ---");
74.         findMinMaxForTeam(allPlayers, "Gabungan");
75.     }
76.
77.     // Helper method untuk mencari min/max satu tim
78.     private void findMinMaxForTeam(ArrayList<Player>
    team, String teamName) {
79.         if (team.isEmpty()) {
80.             System.out.println(teamName + " kosong");
81.             return;
82.         }
83.
84.         // Cari min/max tinggi badan
85.         Player minHeightPlayer = Collections.min(team,
    new HeightComparator(true));
86.         Player maxHeightPlayer = Collections.max(team,
    new HeightComparator(true));
87.
88.         // Cari min/max berat badan
89.         Player minWeightPlayer = Collections.min(team,
    new WeightComparator(true));
90.         Player maxWeightPlayer = Collections.max(team,
    new WeightComparator(true));
91.
92.         System.out.println("Tinggi Badan:");
93.         System.out.println("  Min: " +
    minHeightPlayer.getHeight() + "cm (Player " +
94.
    minHeightPlayer.getPlayerNumber() + " - " +
    minHeightPlayer.getTeam() + ")");
95.         System.out.println("  Max: " +
```

```
   maxHeightPlayer.getHeight() + "cm (Player " +
96.
   maxHeightPlayer.getPlayerNumber() + " - " +
   maxHeightPlayer.getTeam() + ")");
97.
98.         System.out.println("Berat Badan:");
99.         System.out.println("  Min: " +
   minWeightPlayer.getWeight() + "kg (Player " +
100.
   minWeightPlayer.getPlayerNumber() + " - " +
   minWeightPlayer.getTeam() + ")");
101.           System.out.println("  Max: " +
   maxWeightPlayer.getWeight() + "kg (Player " +
102.
   maxWeightPlayer.getPlayerNumber() + " - " +
   maxWeightPlayer.getTeam() + ")");
103.       }
104.
105.       // Helper method untuk menampilkan daftar pemain
106.       private void displayPlayerList(ArrayList<Player>
   players) {
107.           for (Player p : players) {
108.               System.out.println(p);
109.           }
110.       }
111.
112.       // Method tambahan: sorting hanya satu tim
113.       public void sortSingleTeam(String teamName,
   String criteria, boolean ascending) {
114.           ArrayList<Player> team;
115.
116.           switch (teamName.toUpperCase()) {
117.               case "TIM A":
118.               case "A":
119.                   team = dataManager.getTeamA();
120.                   break;
121.               case "TIM B":
122.               case "B":
123.                   team = dataManager.getTeamB();
124.                   break;
125.               case "TIM C":
126.               case "C":
127.                   team = dataManager.getTeamC();
```

```
                    break;
                default:
                    System.out.println("Tim tidak
    ditemukan: " + teamName);
                    return;
            }

            System.out.println("\n--- Sorting " +
    teamName + " berdasarkan " + criteria +
                               (ascending ? " (Ascending)"
    : " (Descending)") + " ---");

            if (criteria.equalsIgnoreCase("height") ||
    criteria.equalsIgnoreCase("tinggi")) {
                Collections.sort(team, new
    HeightComparator(ascending));
            } else if
    (criteria.equalsIgnoreCase("weight") ||
    criteria.equalsIgnoreCase("berat")) {
                Collections.sort(team, new
    WeightComparator(ascending));
            } else {
                System.out.println("Kriteria tidak valid.
    Gunakan 'height' atau 'weight'");
                return;
            }

            displayPlayerList(team);
        }
    }


```

```
// Main.java
// Class utama untuk menjalankan program manajemen tim
    futsal

import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        System.out.println("=== PROGRAM MANAJEMEN TIM
```

```
FUTSAL ===");
9.          System.out.println("Implementasi Collection
Sorting dan Searching");
10.
System.out.println("==================================
========");
11.
12.         // Inisialisasi komponen program
13.         TeamDataManager dataManager = new
TeamDataManager();
14.         SortingOperations sortingOps = new
SortingOperations(dataManager);
15.         SearchOperations searchOps = new
SearchOperations(dataManager);
16.
17.         // Tampilkan data awal
18.         System.out.println("\n1. MENAMPILKAN DATA
AWAL");
19.         dataManager.displayAllTeams();
20.
21.         // ===== BAGIAN 1: SORTING DAN OPERASI
COLLECTION =====
22.         System.out.println("\n" + "=".repeat(60));
23.         System.out.println("BAGIAN 1: SORTING DAN
OPERASI COLLECTION");
24.         System.out.println("=".repeat(60));
25.
26.         // 1a. Sorting berdasarkan tinggi badan
27.         System.out.println("\n1a. SORTING BERDASARKAN
TINGGI BADAN");
28.         sortingOps.sortByHeight();
29.
30.         // 1b. Sorting berdasarkan berat badan
31.         System.out.println("\n1b. SORTING BERDASARKAN
BERAT BADAN");
32.         sortingOps.sortByWeight();
33.
34.         // 1c. Nilai min dan max
35.         System.out.println("\n1c. NILAI MIN DAN MAX
TINGGI BADAN & BERAT BADAN");
36.         sortingOps.findMinMaxValues();
37.
38.         // Demonstrasi fitur tambahan (copy Tim B ke Tim
```

```
   C, sorting per tim, dll)
39.         System.out.println("\n" + "=".repeat(60));
40.         System.out.println("DEMONSTRASI FITUR
   TAMBAHAN");
41.         System.out.println("=".repeat(60));
42.
43.         // Copy Tim B -> Tim C
44.         System.out.println("\nMenyalin Tim B ke Tim
   C...");
45.         dataManager.copyTeamBToTeamC();
46.         dataManager.displayTeam("Tim C");
47.
48.         // Contoh sorting satu tim: Tim A berdasarkan
   tinggi (descending)
49.         System.out.println("\nContoh sorting Tim A
   berdasarkan tinggi (descending):");
50.         sortingOps.sortSingleTeam("Tim A", "height",
   false);
51.
52.         // Tampilkan ukuran tim
53.         dataManager.displayTeamSizes();
54.
55.         // ===== BAGIAN 2: BINARY SEARCH PADA ARRAYLIST
   =====
56.         System.out.println("\n" + "=".repeat(60));
57.         System.out.println("BAGIAN 2: BINARY SEARCH PADA
   ARRAYLIST");
58.         System.out.println("=".repeat(60));
59.
60.         // 2a. Tampilkan jumlah pemain per tim
61.         System.out.println("2a. Jumlah pemain per
   tim:");
62.         System.out.println("    Tim A: " +
   dataManager.getTeamA().size() + " pemain");
63.         System.out.println("    Tim B: " +
   dataManager.getTeamB().size() + " pemain");
64.
65.         // 2b. Cari tinggi 168cm dan 160cm di Tim B
66.         searchOps.searchHeightInTeamB();
67.
68.         // 2c. Cari berat 56kg dan 53kg di Tim A
69.         searchOps.searchWeightInTeamA();
70.
```

```
        // 2d. Cek apakah ada pemain di Tim A yang tingginya 168cm
        int cekHeight = 168;
        boolean ada168 = searchOps.existsHeightInTeamA(cekHeight);
        System.out.printf("\n2d. Apakah ada pemain di Tim A dengan tinggi %d cm? %s%n",
                          cekHeight, (ada168 ? "Ada" : "Tidak ada"));
    }
}
```

```
// SearchOperations.java
// Class untuk menangani operasi pencarian (binary search) dan analisis data tim

import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;
import java.util.Set;

public class SearchOperations {
    private TeamDataManager dataManager;

    public SearchOperations(TeamDataManager dataManager) {
        this.dataManager = dataManager;
    }

    /**
     * 2b. Binary search untuk Tim B berdasarkan tinggi badan:
     *     - Menampilkan list tinggi badan Tim B yang sudah di-sort.
     *     - Menghitung jumlah kemunculan untuk 168 cm dan 160 cm.
     *     - Memberikan hasil binarySearch (found index / insertion point).
     */
    public void searchHeightInTeamB() {
        System.out.println("\n=== BINARY SEARCH TINGGI BADAN TIM B ===");

```

```
25.         ArrayList<Player> teamB =
    dataManager.getTeamB();
26.
27.         // Buat list berisi semua nilai height Tim B
    lalu sort ascending
28.         ArrayList<Integer> heightsB = new ArrayList<>();
29.         for (Player p : teamB) {
30.             heightsB.add(p.getHeight());
31.         }
32.         Collections.sort(heightsB);
33.
34.         System.out.println("Tinggi badan Tim B (sorted):
    " + heightsB);
35.
36.         // Cari tinggi 168 cm
37.         searchSpecificHeight(heightsB, 168);
38.
39.         // Cari tinggi 160 cm
40.         searchSpecificHeight(heightsB, 160);
41.     }
42.
43.     /**
44.      * 2c. Binary search untuk Tim A berdasarkan berat
    badan:
45.      *     - Menampilkan list berat badan Tim A yang
    sudah di-sort.
46.      *     - Menghitung jumlah kemunculan untuk 56 kg
    dan 53 kg.
47.      *     - Memberikan hasil binarySearch (found index
    / insertion point).
48.      */
49.     public void searchWeightInTeamA() {
50.         System.out.println("\n=== BINARY SEARCH BERAT
    BADAN TIM A ===");
51.
52.         ArrayList<Player> teamA =
    dataManager.getTeamA();
53.
54.         // Buat list berisi semua nilai weight Tim A
    lalu sort ascending
55.         ArrayList<Integer> weightsA = new ArrayList<>();
56.         for (Player p : teamA) {
57.             weightsA.add(p.getWeight());
```

```
        }
        Collections.sort(weightsA);

        System.out.println("Berat badan Tim A (sorted): " + weightsA);

        // Cari berat 56 kg
        searchSpecificWeight(weightsA, 56);

        // Cari berat 53 kg
        searchSpecificWeight(weightsA, 53);
    }

    // Helper method untuk mencari jumlah dan posisi tinggi tertentu
    private void searchSpecificHeight(ArrayList<Integer> heights, int targetHeight) {
        int count = Collections.frequency(heights, targetHeight);
        int index = Collections.binarySearch(heights, targetHeight);

        System.out.println("\nTinggi " + targetHeight + "cm:");
        System.out.println("- Jumlah pemain: " + count);
        System.out.println("- Binary search result: " +
                (index >= 0 ? "Found at index " + index : "Not found (" + index + ")"));

        if (index >= 0) {
            System.out.println("- Penjelasan: Ditemukan di posisi indeks " + index +
                    " dalam list yang sudah terurut");
        } else {
            int insertionPoint = -(index + 1);
            System.out.println("- Penjelasan: Tidak ditemukan. Jika ingin disisipkan, " +
                    "posisinya akan berada di indeks " + insertionPoint);
        }
    }

    // Helper method untuk mencari jumlah dan posisi
```

```
    berat tertentu
91.     private void searchSpecificWeight(ArrayList<Integer>
    weights, int targetWeight) {
92.         int count = Collections.frequency(weights,
    targetWeight);
93.         int index = Collections.binarySearch(weights,
    targetWeight);
94.
95.         System.out.println("\nBerat " + targetWeight +
    "kg:");
96.         System.out.println("- Jumlah pemain: " + count);
97.         System.out.println("- Binary search result: " +
98.                 (index >= 0 ? "Found at index " + index
    : "Not found (" + index + ")"));
99.
100.            if (index >= 0) {
101.                System.out.println("- Penjelasan:
    Ditemukan di posisi indeks " + index +
102.                        " dalam list yang sudah
    terurut");
103.            } else {
104.                int insertionPoint = -(index + 1);
105.                System.out.println("- Penjelasan: Tidak
    ditemukan. Jika ingin disisipkan, " +
106.                        "posisinya akan berada di indeks
    " + insertionPoint);
107.            }
108.        }
109.
110.        /**
111.         * 2d. Cek apakah ada pemain di Tim A yang
    mempunyai tinggi = targetHeight.
112.         *     Mengembalikan true jika setidaknya satu
    pemain dengan nilai tersebut ada.
113.         */
114.        public boolean existsHeightInTeamA(int
    targetHeight) {
115.            ArrayList<Player> teamA =
    dataManager.getTeamA();
116.
117.            // Buat list berisi semua height Tim A lalu
    sort ascending
118.            ArrayList<Integer> heightsA = new
```

```
   ArrayList<>();
119.            for (Player p : teamA) {
120.                heightsA.add(p.getHeight());
121.            }
122.            Collections.sort(heightsA);
123.
124.            // Gunakan binarySearch untuk mengecek
   keberadaan
125.            int idx = Collections.binarySearch(heightsA,
   targetHeight);
126.            return idx >= 0;
127.        }
128.
129.        /**
130.         * (Opsional) Fitur tambahan: Cek kesamaan nilai
   tinggi/berat antara Tim A dan Tim B
131.         */
132.        public void checkCommonValues() {
133.            System.out.println("\n=== CEK KESAMAAN DATA
   ANTARA TIM A DAN TIM B ===");
134.
135.            ArrayList<Player> teamA =
   dataManager.getTeamA();
136.            ArrayList<Player> teamB =
   dataManager.getTeamB();
137.
138.            // Kumpulkan nilai tinggi dan berat Tim A ke
   dalam set (unik)
139.            Set<Integer> heightsA = new HashSet<>();
140.            Set<Integer> weightsA = new HashSet<>();
141.            for (Player p : teamA) {
142.                heightsA.add(p.getHeight());
143.                weightsA.add(p.getWeight());
144.            }
145.
146.            // Kumpulkan nilai tinggi dan berat Tim B ke
   dalam set (unik)
147.            Set<Integer> heightsB = new HashSet<>();
148.            Set<Integer> weightsB = new HashSet<>();
149.            for (Player p : teamB) {
150.                heightsB.add(p.getHeight());
151.                weightsB.add(p.getWeight());
152.            }
```

```
153.
154.            // Tampilkan data unik tiap tim
155.            System.out.println("\nData unik Tim A:");
156.            System.out.println("- Tinggi badan: " +
    heightsA);
157.            System.out.println("- Berat badan: " +
    weightsA);
158.
159.            System.out.println("\nData unik Tim B:");
160.            System.out.println("- Tinggi badan: " +
    heightsB);
161.            System.out.println("- Berat badan: " +
    weightsB);
162.
163.            // Cek kesamaan tinggi badan
164.            Set<Integer> commonHeights = new
    HashSet<>(heightsA);
165.            commonHeights.retainAll(heightsB);
166.
167.            System.out.println("\n--- Analisis Kesamaan
    Tinggi Badan ---");
168.            if (commonHeights.isEmpty()) {
169.                System.out.println("Tidak ada tinggi
    badan yang sama antara Tim A dan Tim B");
170.            } else {
171.                System.out.println("Tinggi badan yang
    sama: " + commonHeights + " cm");
172.                displayPlayersWithCommonHeights(teamA,
    teamB, commonHeights);
173.            }
174.
175.            // Cek kesamaan berat badan
176.            Set<Integer> commonWeights = new
    HashSet<>(weightsA);
177.            commonWeights.retainAll(weightsB);
178.
179.            System.out.println("\n--- Analisis Kesamaan
    Berat Badan ---");
180.            if (commonWeights.isEmpty()) {
181.                System.out.println("Tidak ada berat badan
    yang sama antara Tim A dan Tim B");
182.            } else {
183.                System.out.println("Berat badan yang
```

```
   sama: " + commonWeights + " kg");
184.               displayPlayersWithCommonWeights(teamA,
   teamB, commonWeights);
185.           }
186.
187.           // Validasi menggunakan
   Collections.disjoint()
188.           ArrayList<Integer> listHeightsA = new
   ArrayList<>(heightsA);
189.           ArrayList<Integer> listHeightsB = new
   ArrayList<>(heightsB);
190.           ArrayList<Integer> listWeightsA = new
   ArrayList<>(weightsA);
191.           ArrayList<Integer> listWeightsB = new
   ArrayList<>(weightsB);
192.
193.           boolean heightDisjoint =
   Collections.disjoint(listHeightsA, listHeightsB);
194.           boolean weightDisjoint =
   Collections.disjoint(listWeightsA, listWeightsB);
195.
196.           System.out.println("\n--- Validasi dengan
   Collections.disjoint() ---");
197.           System.out.println("Tinggi badan disjoint
   (tidak ada yang sama): " + heightDisjoint);
198.           System.out.println("Berat badan disjoint
   (tidak ada yang sama): " + weightDisjoint);
199.       }
200.
201.       // Helper: tampilkan daftar pemain (Tim A vs Tim
   B) untuk setiap tinggi yang sama
202.       private void
   displayPlayersWithCommonHeights(ArrayList<Player> teamA,
203.
   ArrayList<Player> teamB,
204.
   Set<Integer> commonHeights) {
205.           for (Integer height : commonHeights) {
206.               System.out.println("\nPemain dengan
   tinggi " + height + "cm:");
207.               System.out.println("  Tim A:");
208.               for (Player p : teamA) {
209.                   if (p.getHeight() == height) {
```

```
                    System.out.println("    " + p);
                }
            }
            System.out.println("  Tim B:");
            for (Player p : teamB) {
                if (p.getHeight() == height) {
                    System.out.println("    " + p);
                }
            }
        }
    }

    // Helper: tampilkan daftar pemain (Tim A vs Tim B) untuk setiap berat yang sama
    private void displayPlayersWithCommonWeights(ArrayList<Player> teamA,
ArrayList<Player> teamB,
Set<Integer> commonWeights) {
        for (Integer weight : commonWeights) {
            System.out.println("\nPemain dengan berat " + weight + "kg:");
            System.out.println("  Tim A:");
            for (Player p : teamA) {
                if (p.getWeight() == weight) {
                    System.out.println("    " + p);
                }
            }
            System.out.println("  Tim B:");
            for (Player p : teamB) {
                if (p.getWeight() == weight) {
                    System.out.println("    " + p);
                }
            }
        }
    }

    /**
     * (Opsional) Fitur tambahan: pencarian kustom pada satu tim berdasarkan kriteria 'height' atau 'weight'.
     *
```

```
246.         * @param team     Nama tim ("Tim A", "Tim B",
     atau "Tim C")
247.         * @param criteria "height"/"tinggi" atau
     "weight"/"berat"
248.         * @param value    Nilai yang ingin dicari
     (contoh: 170cm atau 65kg)
249.         */
250.       public void searchCustomValue(String team, String
     criteria, int value) {
251.           ArrayList<Player> targetTeam;
252.
253.           switch (team.toUpperCase()) {
254.               case "TIM A":
255.               case "A":
256.                   targetTeam = dataManager.getTeamA();
257.                   break;
258.               case "TIM B":
259.               case "B":
260.                   targetTeam = dataManager.getTeamB();
261.                   break;
262.               case "TIM C":
263.               case "C":
264.                   targetTeam = dataManager.getTeamC();
265.                   break;
266.               default:
267.                   System.out.println("Tim tidak
     ditemukan: " + team);
268.                   return;
269.           }
270.
271.           // Buat daftar nilai berdasarkan kriteria
272.           ArrayList<Integer> values = new
     ArrayList<>();
273.           if (criteria.equalsIgnoreCase("height") ||
     criteria.equalsIgnoreCase("tinggi")) {
274.               for (Player p : targetTeam) {
275.                   values.add(p.getHeight());
276.               }
277.           } else if
     (criteria.equalsIgnoreCase("weight") ||
     criteria.equalsIgnoreCase("berat")) {
278.               for (Player p : targetTeam) {
279.                   values.add(p.getWeight());
```

```
            }
        } else {
            System.out.println("Kriteria tidak valid. Gunakan 'height' atau 'weight'.");
            return;
        }

        // Sort ascending
        Collections.sort(values);

        int count = Collections.frequency(values, value);
        int index = Collections.binarySearch(values, value);

        System.out.println("\n=== PENCARIAN CUSTOM ===");
        System.out.println("Tim: " + team);
        System.out.println("Kriteria: " + criteria);
        System.out.println("Nilai yang dicari: " + value);
        System.out.println("Jumlah ditemukan: " + count);
        System.out.println("Binary search result: " +
                (index >= 0 ? "Found at index " + index : "Not found (" + index + ")"));
    }
}
```

D. Penjelasan

(Bagian 1: operasi sorting/collection, dan Bagian 2: operasi pencarian/binary search). Penjelasan ini akan membahas:

1. **Deskripsi umum tugas**
2. **Struktur kelas dan model data**
3. **Bagian 1: Sorting dan operasi Collection**

- 1a. Sorting berdasarkan tinggi badan
- 1b. Sorting berdasarkan berat badan
- 1c. Mencari nilai minimum dan maksimum (tinggi/berat)
- 1d. Menyalin anggota Tim B → Tim C dan sorting khusus per tim

4. **Bagian 2: Pencarian (Binary Search) pada ArrayList**
   - 2a. Menyimpan data Tim A dan Tim B dalam ArrayList terpisah
   - 2b. Menghitung jumlah pemain Tim B dengan tinggi 168 cm dan 160 cm via binary search
   - 2c. Menghitung jumlah pemain Tim A dengan berat 56 kg dan 53 kg via binary search
   - 2d. Mengecek apakah ada pemain Tim A dengan tinggi tertentu (contoh: 168 cm)
5. **Penjelasan detail implementasi kode (kelas, metode, alur)**
6. **Cara menjalankan dan contoh output**
7. **Ringkasan poin-poin utama**

1. Deskripsi Umum Tugas

Soal menuntut pembuatan sebuah program Java yang memanajemen data dua tim futsal (Tim A dan Tim B), yang setiap pemainnya memiliki atribut **tinggi badan** (dalam cm) dan **berat badan** (dalam kg). Kita harus mengimplementasikan:

1. **Bagian 1**:
   - a) Urutkan semua pemain (gabungan Tim A + Tim B) berdasarkan tinggi badan (ascending & descending).
   - b) Urutkan semua pemain (gabungan Tim A + Tim B) berdasarkan berat badan (ascending & descending).
   - c) Temukan nilai minimum dan maksimum untuk tinggi badan dan berat badan, **per tim** (Tim A dan Tim B), dan—jika diinginkan—per gabungan (Tim A + Tim B).
   - d) Buat salinan (copy) seluruh anggota Tim B menjadi Tim C, lalu demonstrasikan sorting khusus pada salah satu tim (misal Tim A saja).
2. **Bagian 2**:

- a) Simpan data Tim A dan data Tim B dalam dua buah ArrayList<Player> terpisah (ini menjawab soal “Implementasikan ArrayList untuk menyimpan data tim A dan Tim B dalam bentuk ArrayList terpisah”).

- b) Dari data Tim B, hitung berapa pemain yang mempunyai tinggi badan 168 cm dan berapa yang 160 cm, **menggunakan** (atau minimal mencontoh) konsep binary search (atau metode sejenis).

- c) Dari data Tim A, hitung berapa pemain yang mempunyai berat badan 56 kg dan berapa yang 53 kg, juga dengan binary search.

- d) Cek apakah di Tim A ada pemain dengan tinggi (misal) 168 cm. (Jika ya → cetak “Ada”, jika tidak → “Tidak ada”.)

Dengan kata lain, Bagian 1 lebih menitikberatkan pada operasional sorting dan statistik (min/max, copy-list, sorting individual tim), sedangkan Bagian 2 menitikberatkan pada kemampuan mencari nilai tertentu di dalam data yang sudah di‑sort (binary search, frequency count).

2. Struktur Kelas dan Model Data

Untuk menyelesaikan soal ini, kita membangun beberapa kelas utama:

1. **Player.java**

Model data untuk satu pemain futsal, dengan atribut:

```
    private int height; // tinggi badan (cm)

    private int weight; // berat badan (kg)

    private String team; // nama tim

    private int playerNumber; // nomor pemain
```

Constructor:

public Player(int id, int height, int weight, String teamName) { … }

Getter:

public int getId() { … }

public int getHeight() { … }

public int getWeight() { … }

public String getTeamName() { … }

-

toString() override, misalnya:

```
@Override
public String toString() {
    return String.format("Player %d (%s): Height=%dcm, Weight=%dkg",
                         id, teamName, height, weight);
}
```

-
- Method tambahan copyToTeam(String newTeamName) untuk membuat salinan objek Player dengan nama tim diganti, diperlukan saat menyalin Tim B → Tim C.

2. **HeightComparator.java** & **WeightComparator.java**

Dua buah kelas yang mengimplementasikan Comparator<Player>:

```
public class HeightComparator implements Comparator<Player> {
    @Override
    public int compare(Player p1, Player p2) {
        return Integer.compare(p1.getHeight(), p2.getHeight());
    }
}
public class WeightComparator implements Comparator<Player> {
    @Override
    public int compare(Player p1, Player p2) {
        return Integer.compare(p1.getWeight(), p2.getWeight());
    }
}
```

-
- Digunakan untuk menyortir ArrayList<Player> berdasarkan atribut height atau weight.

3. **TeamDataManager.java**

Kelas ini bertugas **mengelola data semua tim**. Di dalamnya terdapat:

private ArrayList<Player> teamA;

private ArrayList<Player> teamB;

private ArrayList<Player> teamC;

-

Pada constructor, memanggil initializeTeams() yang mengisi teamA dan teamB dari data statis (sesuai soal). Contohnya:

```
int[][] dataTimA = {
    {168,50}, {170,60}, {165,56}, {168,55}, {172,60},
    {170,70}, {169,66}, {165,56}, {171,72}, {166,56}
};
int[][] dataTimB = {
    {170,66}, {167,60}, {165,59}, {166,58}, {168,58},
    {175,71}, {172,68}, {171,68}, {168,65}, {169,60}
};
```

// Lalu for-loop membuat Player(i+1, tinggi, berat, "Tim A") → dimasukkan ke teamA

// dan Player(i+1, tinggi, berat, "Tim B") → dimasukkan ke teamB.

-
- teamC awalnya dikosongkan; nanti di‑isi ketika memanggil copyTeamBToTeamC().
- **Method‑method utama**:

1. public ArrayList<Player> getTeamA(): mengembalikan salinan list teamA.
2. public ArrayList<Player> getTeamB(): mengembalikan salinan list teamB.
3. public ArrayList<Player> getTeamC(): mengembalikan salinan list teamC.
4. public ArrayList<Player> getAllPlayers(): menggabungkan teamA + teamB + (jika sudah ada) teamC ke dalam satu ArrayList<Player> dan mengembalikannya.

```
public void copyTeamBToTeamC():
 teamC.clear();
```

```
for (Player p : teamB) {
    teamC.add(p.copyToTeam("Tim C"));
}
```

5.
6. public void displayTeam(String teamName): mencetak semua daftar pemain sesuai teamName ("Tim A", "Tim B", atau "Tim C") ke console.

7. public void displayAllTeams(): memanggil displayTeam("Tim A") dan displayTeam("Tim B"), lalu jika teamC tidak kosong, juga displayTeam("Tim C").

public void displayTeamSizes(): mencetak:

=== JUMLAH PEMAIN ===

Tim A: X pemain

Tim B: Y pemain

Tim C: Z pemain

8.

4. **SortingOperations.java**

Kelas ini menangani semua operasi sorting (Bagian 1). Di dalamnya terdapat:

```
private TeamDataManager dataManager;
public SortingOperations(TeamDataManager dataManager) {
    this.dataManager = dataManager;
}
```

-
- **Method-method utama**:

1. public void sortByHeight():
    - Mengambil ArrayList<Player> all = dataManager.getAllPlayers().
    - **Ascending**: Collections.sort(all, new HeightComparator()); → cetak "=== SORTING BERDASARKAN TINGGI BADAN (Ascending) ===" lalu for (Player p : all) System.out.println(p);
    - **Descending**: Collections.sort(all, Collections.reverseOrder(new HeightComparator()));

→ cetak “=== SORTING BERDASARKAN TINGGI BADAN (Descending) ===” lalu print.

2. public void sortByWeight():

    - Mirip sortByHeight(), tetapi memakai WeightComparator untuk sort ascending/descending berdasarkan weight.

3. public void findMinMaxValues():

Mengambil teamA = dataManager.getTeamA(). Jika tidak kosong, gunakan Java Stream:

```
Optional<Player> minH_A = teamA.stream().min(Comparator.comparingInt(Player::getHeight));
Optional<Player> maxH_A = teamA.stream().max(Comparator.comparingInt(Player::getHeight));
Optional<Player> minW_A = teamA.stream().min(Comparator.comparingInt(Player::getWeight));
Optional<Player> maxW_A = teamA.stream().max(Comparator.comparingInt(Player::getWeight));
```

→ cetak:

--- Tim A ---

Tinggi Badan:

Min: <nilai>cm (Player <id>)

Max: <nilai>cm (Player <id>)

Berat Badan:

Min: <nilai>kg (Player <id>)

Max: <nilai>kg (Player <id>)

-
- Lakukan hal sama untuk teamB.

(Opsional): Ambil all = dataManager.getAllPlayers() dan cari min/max di gabungan A + B, lalu cetak:

--- Gabungan Tim A & B ---

Tinggi Badan:

Min: … (Player … – Tim X)

Max: … (Player … – Tim Y)

Berat Badan:

Min: … (Player … – Tim X)

Max: … (Player … – Tim Y)

-

4. public void sortSingleTeam(String teamName, String field, boolean ascending):
    - Berdasarkan argumen teamName, ambil list dari dataManager.getTeamA(), getTeamB(), atau getTeamC().
    - Jika field.equalsIgnoreCase("height"), maka Collections.sort(teamList, new HeightComparator()), kemudian jika !ascending → Collections.reverse(teamList).
    - Jika field.equalsIgnoreCase("weight"), maka pakai WeightComparator.

Cetak judul:

--- Sorting <teamName> berdasarkan <tinggi/berat> (<Ascending/Descending>) ---

- Lalu print seluruh pembaca dari teamList.

5. **SearchOperations.java**

Kelas ini khusus untuk Bagian 2 (binary search & analisis tambahan). Isinya:

```
private TeamDataManager dataManager;

public SearchOperations(TeamDataManager dataManager) {
    this.dataManager = dataManager;
}
```

-
- **Method-method utama**:

1. public void searchHeightInTeamB():
    - Cetak header “=== BINARY SEARCH TINGGI BADAN TIM B ===”.
    - Ambil teamB = dataManager.getTeamB().
    - Konversi ke ArrayList<Integer> heightsB = new ArrayList<>(), lalu buat loop for (Player p : teamB) heightsB.add(p.getHeight());

- Collections.sort(heightsB); → cetak Tinggi badan Tim B (sorted): [....].
- Panggil helper searchSpecificHeight(heightsB, 168);
- Panggil searchSpecificHeight(heightsB, 160);
- **Helper searchSpecificHeight(ArrayList<Integer> heights, int targetHeight)**:
  - int count = Collections.frequency(heights, targetHeight); → ini menghitung berapa kali targetHeight muncul di heights.
  - int index = Collections.binarySearch(heights, targetHeight);
    - Jika index >= 0, berarti ada pemain dengan tinggi targetHeight di posisi index.
    - Jika index < 0, binarySearch mengembalikan -(insertionPoint) - 1. Misal, jika ingin disisipkan, insertionPoint = -(index+1).

Cetak:

Tinggi <targetHeight>cm:

- Jumlah pemain: <count>

- Binary search result:

(jika index>=0) "Found at index <index>"

(jika index<0) "Not found (<index>)"

- Penjelasan:

Jika index>=0 → "Ditemukan di posisi indeks <index> dalam list yang sudah terurut"

Jika index<0 → "Tidak ditemukan. Jika ingin disisipkan, posisinya akan berada di indeks <insertionPoint>"

- 2. public void searchWeightInTeamA():
  - Sama seperti searchHeightInTeamB(), tetapi untuk ArrayList<Integer> weightsA = … yang di‑isi dari p.getWeight() per Player p pada teamA.
  - Cetak “=== BINARY SEARCH BERAT BADAN TIM A ===” dan hasil sort “Berat badan Tim A (sorted): […]”.

- Panggil searchSpecificWeight(weightsA, 56); dan searchSpecificWeight(weightsA, 53);.

- **Helper searchSpecificWeight(ArrayList<Integer> weights, int targetWeight)**: fungsi identik dengan searchSpecificHeight tetapi cuma mengganti label “Berat” dan targetWeight.

3. public boolean existsHeightInTeamA(int targetHeight):

    - Ambil teamA = dataManager.getTeamA().

    - Buat ArrayList<Integer> heightsA = new ArrayList<>(), loop isi dengan p.getHeight().

    - Collections.sort(heightsA);

    - int idx = Collections.binarySearch(heightsA, targetHeight);

    - Return idx >= 0. Jika ada, berarti minimal satu pemain Tim A memiliki tinggi targetHeight.

4. **Opsional (tidak wajib soal, tapi sering disediakan sebagai fitur tambahan):**

    - public void checkCommonValues():

        - Mengumpulkan nilai tinggi dan berat teamA ke dalam Set<Integer> heightsA dan Set<Integer> weightsA agar unik.

        - Mengumpulkan nilai tinggi dan berat teamB ke dalam Set<Integer> heightsB dan Set<Integer> weightsB.

        - Cetak “Data unik Tim A: …” dan “Data unik Tim B: …” (hanya set).

        - Temukan irisan (intersection) commonHeights = new HashSet<>(heightsA); commonHeights.retainAll(heightsB);. Jika tidak kosong, cetak nilai-nilai tinggi yang sama, lalu loop masing-masing tim menampilkan daftar Player yang memiliki tinggi tersebut.

        - Lakukan serupa untuk commonWeights.

        - Sebagai validasi akhir, bisa gunakan Collections.disjoint(listHeightsA, listHeightsB) untuk mengecek apakah benar-benar tidak ada elemen yang sama.

Singkatnya, setiap kelas memiliki tanggung jawab yang terpisah:

- **Player**: model data pemain
- **HeightComparator/WeightComparator**: cara membandingkan untuk sorting
- **TeamDataManager**: penyimpan dan pengelola data tim: inisialisasi, getter, copy (Tim B→Tim C), display semua tim, display jumlah
- **SortingOperations**: semua kebutuhan Bagian 1: sorting gabungan, min/max, sorting individual
- **SearchOperations**: semua kebutuhan Bagian 2: binary search pada ArrayList setelah menyort data, dan fitur tambahan (jika diinginkan)

Terakhir, **Main.java** menjadi entry point yang secara berurutan memanggil:

1. TeamDataManager dataManager = new TeamDataManager();
2. SortingOperations sortingOps = new SortingOperations(dataManager);
3. SearchOperations searchOps = new SearchOperations(dataManager);
4. Cetak judul program
5. Tampilkan data awal → dataManager.displayAllTeams();
6. Bagian 1:
    - sortingOps.sortByHeight();
    - sortingOps.sortByWeight();
    - sortingOps.findMinMaxValues();
    - Copy Tim B → Tim C: dataManager.copyTeamBToTeamC(); dataManager.displayTeam("Tim C");
    - Contoh sorting khusus: sortingOps.sortSingleTeam("Tim A","height", false);
    - Tampilkan jumlah pemain tiap tim: dataManager.displayTeamSizes();
7. Bagian 2:
    - Cetak “2a. Jumlah pemain per tim: Tim A: … pemain, Tim B: … pemain”
    - Panggil searchOps.searchHeightInTeamB();
    - Panggil searchOps.searchWeightInTeamA();

- Panggil searchOps.existsHeightInTeamA(168); untuk menampilkan “Ada”/“Tidak ada”

3. Bagian 1: Sorting dan Operasi Collection

3.1. Data awal

Sebelum bagian 1 dijalankan, di TeamDataManager.initializeTeams() sudah dimuat data statis:

Tim A (10 pemain):

{168,50}, {170,60}, {165,56}, {168,55}, {172,60},

{170,70}, {169,66}, {165,56}, {171,72}, {166,56}

→ Membentuk 10 objek Player:

Player 1 (Tim A): Height=168, Weight=50

Player 2 (Tim A): Height=170, Weight=60

Player 3 (Tim A): Height=165, Weight=56

… dst …

-

Tim B (10 pemain):

{170,66}, {167,60}, {165,59}, {166,58}, {168,58},

{175,71}, {172,68}, {171,68}, {168,65}, {169,60}

- → Membentuk 10 objek Player di teamB.

Awalnya teamC kosong; baru akan di‑populate ketika diperlukan.

3.2. 1a. Sorting Berdasarkan Tinggi Badan

Langkah‑langkah

1. **Ambil semua pemain** (Tim A + Tim B + Tim C jika ada) dengan memanggil dataManager.getAllPlayers().
   Karena saat pertama kali dijalankan teamC masih kosong, maka getAllPlayers() hanya menggabungkan teamA (10 pemain) dan teamB (10 pemain) → total 20 pemain.

Sort ascending berdasarkan tinggi:

Collections.sort(all, new HeightComparator());

HeightComparator.compare(p1,p2) membandingkan p1.getHeight() vs p2.getHeight().
Setelah itu, all terurut dari nilai height paling kecil ke terbesar. Contoh urutan ascending (berdasarkan contoh data di soal):

(165,56) Tim A – Player 3

(165,56) Tim A – Player 8

(165,59) Tim B – Player 3

(166,56) Tim A – Player 10

(166,58) Tim B – Player 4

(167,60) Tim B – Player 2

(168,50) Tim A – Player 1

(168,55) Tim A – Player 4

(168,58) Tim B – Player 5

(168,65) Tim B – Player 9

(169,66) Tim A – Player 7

(169,60) Tim B – Player 10

(170,60) Tim A – Player 2

(170,70) Tim A – Player 6

(170,66) Tim B – Player 1

(171,72) Tim A – Player 9

(171,68) Tim B – Player 8

(172,60) Tim A – Player 5

(172,68) Tim B – Player 7

(175,71) Tim B – Player 6

2. **Cetak** judul dan setiap Player p : all dengan System.out.println(p).

Sort descending:

Collections.sort(all, Collections.reverseOrder(new HeightComparator()));

3. Sekarang daftar all terbalik: dari tinggi terbesar ke terendah. Cetak dengan judul "=== SORTING BERDASARKAN TINGGI BADAN (Descending) ===".

Contoh output Bagian 1a

=== SORTING BERDASARKAN TINGGI BADAN (Ascending) ===

Player 3 (Tim A): Height=165cm, Weight=56kg

Player 8 (Tim A): Height=165cm, Weight=56kg

Player 3 (Tim B): Height=165cm, Weight=59kg

Player 10 (Tim A): Height=166cm, Weight=56kg

Player 4 (Tim B): Height=166cm, Weight=58kg

Player 2 (Tim B): Height=167cm, Weight=60kg

Player 1 (Tim A): Height=168cm, Weight=50kg

Player 4 (Tim A): Height=168cm, Weight=55kg

Player 5 (Tim B): Height=168cm, Weight=58kg

Player 9 (Tim B): Height=168cm, Weight=65kg

Player 7 (Tim A): Height=169cm, Weight=66kg

Player 10 (Tim B): Height=169cm, Weight=60kg

Player 2 (Tim A): Height=170cm, Weight=60kg

Player 6 (Tim A): Height=170cm, Weight=70kg

Player 1 (Tim B): Height=170cm, Weight=66kg

Player 9 (Tim A): Height=171cm, Weight=72kg

Player 8 (Tim B): Height=171cm, Weight=68kg

Player 5 (Tim A): Height=172cm, Weight=60kg

Player 7 (Tim B): Height=172cm, Weight=68kg

Player 6 (Tim B): Height=175cm, Weight=71kg

=== SORTING BERDASARKAN TINGGI BADAN (Descending) ===

Player 6 (Tim B): Height=175cm, Weight=71kg

Player 5 (Tim A): Height=172cm, Weight=60kg

Player 7 (Tim B): Height=172cm, Weight=68kg

Player 9 (Tim A): Height=171cm, Weight=72kg

Player 8 (Tim B): Height=171cm, Weight=68kg

Player 2 (Tim A): Height=170cm, Weight=60kg

Player 6 (Tim A): Height=170cm, Weight=70kg

Player 1 (Tim B): Height=170cm, Weight=66kg

Player 7 (Tim A): Height=169cm, Weight=66kg

Player 10 (Tim B): Height=169cm, Weight=60kg

Player 1 (Tim A): Height=168cm, Weight=50kg

Player 4 (Tim A): Height=168cm, Weight=55kg

Player 5 (Tim B): Height=168cm, Weight=58kg

Player 9 (Tim B): Height=168cm, Weight=65kg

Player 2 (Tim B): Height=167cm, Weight=60kg

Player 10 (Tim A): Height=166cm, Weight=56kg

Player 4 (Tim B): Height=166cm, Weight=58kg

Player 3 (Tim A): Height=165cm, Weight=56kg

Player 8 (Tim A): Height=165cm, Weight=56kg

Player 3 (Tim B): Height=165cm, Weight=59kg

3.3. 1b. Sorting Berdasarkan Berat Badan

Prinsipnya sama dengan 1a, hanya saja menggunakan WeightComparator untuk membandingkan p.getWeight():

1. **Ambil** ArrayList<Player> all = dataManager.getAllPlayers() (masih 20 pemain).

2. **Sort ascending** dengan Collections.sort(all, new WeightComparator()). Cetak "=== SORTING BERDASARKAN BERAT BADAN (Ascending) ===" → print setiap p.

3. **Sort descending** dengan Collections.reverseOrder(new WeightComparator()). Cetak "=== SORTING BERDASARKAN BERAT BADAN (Descending) ===" → print.

Contoh urutan ascending (berdasarkan contoh data):

Player 1 (Tim A): Height=168cm, Weight=50kg

Player 4 (Tim A): Height=168cm, Weight=55kg

Player 3 (Tim A): Height=165cm, Weight=56kg

Player 8 (Tim A): Height=165cm, Weight=56kg

Player 10 (Tim A): Height=166cm, Weight=56kg

Player 4 (Tim B): Height=166cm, Weight=58kg

Player 5 (Tim B): Height=168cm, Weight=58kg

Player 3 (Tim B): Height=165cm, Weight=59kg

Player 2 (Tim A): Height=170cm, Weight=60kg

Player 5 (Tim A): Height=172cm, Weight=60kg

Player 2 (Tim B): Height=167cm, Weight=60kg

Player 10 (Tim B): Height=169cm, Weight=60kg

Player 9 (Tim B): Height=168cm, Weight=65kg

Player 7 (Tim A): Height=169cm, Weight=66kg

Player 1 (Tim B): Height=170cm, Weight=66kg

Player 7 (Tim B): Height=172cm, Weight=68kg

Player 8 (Tim B): Height=171cm, Weight=68kg

Player 6 (Tim A): Height=170cm, Weight=70kg

Player 6 (Tim B): Height=175cm, Weight=71kg

Player 9 (Tim A): Height=171cm, Weight=72kg

Contoh urutan descending:

Player 9 (Tim A): Height=171cm, Weight=72kg

Player 6 (Tim B): Height=175cm, Weight=71kg

Player 6 (Tim A): Height=170cm, Weight=70kg

Player 7 (Tim B): Height=172cm, Weight=68kg

Player 8 (Tim B): Height=171cm, Weight=68kg

Player 7 (Tim A): Height=169cm, Weight=66kg

Player 1 (Tim B): Height=170cm, Weight=66kg

Player 9 (Tim B): Height=168cm, Weight=65kg

Player 2 (Tim A): Height=170cm, Weight=60kg

Player 5 (Tim A): Height=172cm, Weight=60kg

Player 2 (Tim B): Height=167cm, Weight=60kg

Player 10 (Tim B): Height=169cm, Weight=60kg

Player 3 (Tim B): Height=165cm, Weight=59kg

Player 4 (Tim B): Height=166cm, Weight=58kg

Player 5 (Tim B): Height=168cm, Weight=58kg

Player 3 (Tim A): Height=165cm, Weight=56kg

Player 8 (Tim A): Height=165cm, Weight=56kg

Player 10 (Tim A): Height=166cm, Weight=56kg

Player 4 (Tim A): Height=168cm, Weight=55kg

Player 1 (Tim A): Height=168cm, Weight=50kg

3.4. 1c. Nilai Minimum & Maksimum (Tinggi/ Berat)

Maksud soal

- Cari nilai terendah (min) dan tertinggi (max) dari atribut height dan weight, **untuk tiap tim**.
- (Opsional) Cari juga min dan max di gabungan (Tim A + Tim B).

Cara implementasi

- **Per tim** (Tim A dan Tim B), kita sudah punya ArrayList<Player> teamA dan teamB.

Gunakan Java 8 Stream:

```
Optional<Player> minH_A = teamA.stream().min(Comparator.comparingInt(Player::getHeight));
Optional<Player> maxH_A = teamA.stream().max(Comparator.comparingInt(Player::getHeight));
Optional<Player> minW_A = teamA.stream().min(Comparator.comparingInt(Player::getWeight));
Optional<Player> maxW_A = teamA.stream().max(Comparator.comparingInt(Player::getWeight));
```

Kemudian cetak:

```
--- Tim A ---
```

Tinggi Badan:

Min: <minH_A.get().getHeight()>cm (Player <minH_A.get().getId()>)

Max: <maxH_A.get().getHeight()>cm (Player <maxH_A.get().getId()>)

Berat Badan:

Min: <minW_A.get().getWeight()>kg (Player <minW_A.get().getId()>)

Max: <maxW_A.get().getWeight()>kg (Player <maxW_A.get().getId()>)

- Lakukan sama untuk Tim B.

(Opsional):

```
ArrayList<Player> all = dataManager.getAllPlayers();
Optional<Player> minH_all = all.stream().min(Comparator.comparingInt(Player::getHeight));
Optional<Player> maxH_all = all.stream().max(Comparator.comparingInt(Player::getHeight));
Optional<Player> minW_all = all.stream().min(Comparator.comparingInt(Player::getWeight));
Optional<Player> maxW_all = all.stream().max(Comparator.comparingInt(Player::getWeight));
```

- Cetak serupa untuk “Gabungan Tim A & B”.

Contoh nilai min/max (berdasarkan data di soal)

- **Tim A**:
    - Tinggi:
        - Min = 165 cm (Player 3 atau Player 8, tetapi Stream memilih salah satu—biasanya Player 3)
        - Max = 172 cm (Player 5)
    - Berat:
        - Min = 50 kg (Player 1)
        - Max = 72 kg (Player 9)
- **Tim B**:
    - Tinggi:
        - Min = 165 cm (Player 3)

        - Max = 175 cm (Player 6)
    - Berat:
        - Min = 58 kg (Player 4 dan Player 5—Stream memilih Player 4)
        - Max = 71 kg (Player 6)
- **Gabungan A + B (opsional)**:
    - Tinggi:
        - Min = 165 cm (Player 3 dari Tim A)
        - Max = 175 cm (Player 6 dari Tim B)
    - Berat:
        - Min = 50 kg (Player 1 dari Tim A)
        - Max = 72 kg (Player 9 dari Tim A)

3.5. 1d. Copy Tim B → Tim C & Sorting Individual Tim

Menyalin Tim B ke Tim C

Di TeamDataManager sudah ada method:

```
public void copyTeamBToTeamC() {
    teamC.clear();
    for (Player p : teamB) {
        teamC.add(p.copyToTeam("Tim C"));
    }
}
```

- p.copyToTeam("Tim C") membuat new Player(id, height, weight, "Tim C").
- Setelah eksekusi, teamC berisi 10 pemain—tepat sama datanya dengan Tim B, hanya teamName diganti “Tim C”.

Di Main.java, kita panggil:

```
 dataManager.copyTeamBToTeamC();
dataManager.displayTeam("Tim C");
```

-

Sorting Individual Tim

Di soal Bagian 1d disebut “Contoh sorting Tim A berdasarkan tinggi (descending)”. Kita implementasikan di SortingOperations.sortSingleTeam(String teamName, String field, boolean ascending):

1. **Tentukan tim** berdasarkan teamName (“Tim A” atau “A” mengembalikan getTeamA(), “Tim B” atau “B” → getTeamB(), “Tim C”/“C” → getTeamC()).

2. Jika list kosong, cetak “Tim kosong” lalu return.

Cetak judul, misalnya:

--- Sorting Tim A berdasarkan tinggi (Descending) ---

```
Jika field.equalsIgnoreCase("height") →

 Collections.sort(teamList, new HeightComparator());

if (!ascending) {

    Collections.reverse(teamList);

}
```

Jika field.equalsIgnoreCase("weight") →

```
 Collections.sort(teamList, new WeightComparator());

if (!ascending) {

    Collections.reverse(teamList);

}
```

Print semua Player di list tersebut.

4. Bagian 2: Pencarian (Binary Search) pada ArrayList

4.1. 2a. Menyimpan Data Tim A dan Tim B di Dua ArrayList

Meskipun data sudah tersimpan di TeamDataManager sebagai teamA dan teamB, Bagian 2 mengharuskan kita “menunjukkan” bahwa kita menggunakan dua buah ArrayList<Player> terpisah. Di Main.java, caranya sederhana:

```
System.out.println("2a. Jumlah pemain per tim:");

System.out.println("    Tim A: " + dataManager.getTeamA().size() + " pemain");
```

System.out.println("   Tim B: " + dataManager.getTeamB().size() + " pemain");

- dataManager.getTeamA() mengembalikan **copy** ArrayList<Player> untuk Tim A.
- dataManager.getTeamB() mengembalikan **copy** ArrayList<Player> untuk Tim B.

Dengan ini, sebetulnya kita sudah "memiliki" dua ArrayList<Player> terpisah—1 untuk Tim A dan 1 untuk Tim B.

4.2. 2b. Binary Search untuk Tim B → Menghitung Pemain Tinggi 168 cm & 160 cm

Tujuan

- Cari berapa pemain Tim B yang tingginya = 168 cm, dan berapa yang 160 cm.
- Tampilkan hasil binary search (apakah ditemukan, di indeks berapa; jika tidak, di mana insertion point‑nya).

Algoritma

1. Ambil ArrayList<Player> teamB = dataManager.getTeamB().

Ekstrak ke ArrayList<Integer> heightsB:

```
ArrayList<Integer> heightsB = new ArrayList<>();
for (Player p : teamB) {
    heightsB.add(p.getHeight());
}
```

2.
3. Urutkan ascending: Collections.sort(heightsB);.
4. Print Tinggi badan Tim B (sorted): […].
5. Panggil helper method searchSpecificHeight(heightsB, 168); →
    - count = Collections.frequency(heightsB, 168);
    - index = Collections.binarySearch(heightsB, 168);
    - Jika index >= 0, berarti "Found at index …" dan jelaskan "Ditemukan di posisi indeks … dalam list yang sudah terurut".

- Jika index < 0, “Not found (index)”, insertion point = -(index + 1).

6. Panggil searchSpecificHeight(heightsB, 160); (serupa).

Karena di data Tim B terdapat **dua** entri “168” (pada baris {168,58} dan {168,65}), maka:

Setelah sort ascending, heightsB =

[165, 166, 167, 168, 168, 168, 169, 170, 171, 172, 175]

- (Seharusnya hanya ada dua 168, bukan tiga—perhatikan contoh data Tim B yang diberikan hanya 10 elemen: {168,58} dan {168,65}. Sehingga listnya: [165,166,167,168,168,169,170,171,172,175]).
- Collections.frequency(heightsB, 168) akan memberi “2” karena ada dua elemen “168”.
- Misal Collections.binarySearch(...) mengembalikan index = 3 (posisi kemunculan pertamanya; library tak menjamin indeks mana—yang penting >= 0).
- Untuk “160”, frequency = 0, binarySearch akan mengembalikan negative (biasanya –1).

Contoh output Bagian 2b

=== BINARY SEARCH TINGGI BADAN TIM B ===

Tinggi badan Tim B (sorted): [165, 166, 167, 168, 168, 169, 170, 171, 172, 175]

Tinggi 168cm:

- Jumlah pemain: 2

- Binary search result: Found at index 3

- Penjelasan: Ditemukan di posisi indeks 3 dalam list yang sudah terurut

Tinggi 160cm:

- Jumlah pemain: 0

- Binary search result: Not found (-1)

- Penjelasan: Tidak ditemukan. Jika ingin disisipkan, posisinya akan berada di indeks 0

4.3. 2c. Binary Search untuk Tim A → Menghitung Pemain Berat 56 kg & 53 kg

Tujuan

- Cari berapa pemain Tim A yang beratnya = 56 kg, dan berapa yang 53 kg.
- Tampilkan hasil binary search (found/not found dan indeks/insertion point).

Algoritma

1. Ambil ArrayList<Player> teamA = dataManager.getTeamA().

Ekstrak ke ArrayList<Integer> weightsA:

```
ArrayList<Integer> weightsA = new ArrayList<>();
for (Player p : teamA) {
    weightsA.add(p.getWeight());
}
```

2. Sort ascending: Collections.sort(weightsA);.
3. Print Berat badan Tim A (sorted): […].
4. Panggil searchSpecificWeight(weightsA, 56); dan searchSpecificWeight(weightsA, 53);.
    - count = Collections.frequency(weightsA, 56) (ada **tiga** pemain 56: Player 3, Player 8, Player 10).
    - binarySearch(...) kemungkinan mengembalikan index = 2 (atau salah satu dari ketiganya).
    - Untuk 53 kg, count = 0, index = –1.

Contoh output Bagian 2c

=== BINARY SEARCH BERAT BADAN TIM A ===

Berat badan Tim A (sorted): [50, 55, 56, 56, 56, 60, 60, 66, 70, 72]

Berat 56kg:

- Jumlah pemain: 3

- Binary search result: Found at index 2

- Penjelasan: Ditemukan di posisi indeks 2 dalam list yang sudah terurut

Berat 53kg:

- Jumlah pemain: 0

- Binary search result: Not found (-1)

- Penjelasan: Tidak ditemukan. Jika ingin disisipkan, posisinya akan berada di indeks 2

4.4. 2d. Mengecek Keberadaan Pemain di Tim A dengan Tinggi Tertentu

Tujuan

- Ingin mengetahui: apakah ada minimal satu pemain di Tim A yang tinggi badannya = 168 cm? Cetak “Ada” jika ya, “Tidak ada” jika tidak.

Cara

- Panggil boolean exists = searchOps.existsHeightInTeamA(168);

Implementasi existsHeightInTeamA(int targetHeight):

```
public boolean existsHeightInTeamA(int targetHeight) {
    ArrayList<Player> teamA = dataManager.getTeamA();
    ArrayList<Integer> heightsA = new ArrayList<>();
    for (Player p : teamA) {
        heightsA.add(p.getHeight());
    }
    Collections.sort(heightsA);
    int idx = Collections.binarySearch(heightsA, targetHeight);
    return idx >= 0;
}
```

-

Jika dikembalikan true, cetak:

2d. Apakah ada pemain di Tim A dengan tinggi 168 cm? Ada

- Jika false, cetak “Tidak ada”.

Note

- Sebenarnya Anda bisa memanfaatkan searchSpecificHeight(…) dari Bagian 2b—ketika count > 0, otomatis ada. Tetapi membuat method existsHeightInTeamA(...) secara eksplisit memisahkan logika “cek yes/no”.

5. Alur Implementasi Kode Lengkap

Berikut ini ringkasan urutan file‑file penting, urutan eksekusi, dan bagaimana semua berinteraksi.

Player.java

```
public class Player {
    private int id;
    private int height;
    private int weight;
    private String teamName;
    public Player(int id, int height, int weight, String teamName) { … }
    public int getId() { return id; }
    public int getHeight() { return height; }
    public int getWeight() { return weight; }
    public String getTeamName() { return teamName; }
    public Player copyToTeam(String newTeamName) {
        return new Player(this.id, this.height, this.weight, newTeamName);
    }
    @Override
    public String toString() {
        return String.format("Player %d (%s): Height=%dcm, Weight=%dkg",
                             id, teamName, height, weight);
    }
}
```

1. **HeightComparator.java** & **WeightComparator.java**

```
public class HeightComparator implements Comparator<Player> {
    @Override
    public int compare(Player p1, Player p2) {
        return Integer.compare(p1.getHeight(), p2.getHeight());
    }
}
public class WeightComparator implements Comparator<Player> {
    @Override
    public int compare(Player p1, Player p2) {
        return Integer.compare(p1.getWeight(), p2.getWeight());
    }
}
```

2. **TeamDataManager.java**
    - Memuat data statis Tim A + Tim B di initializeTeams().
    - Menyediakan getter getTeamA(), getTeamB(), getTeamC(), getAllPlayers().
    - Method copyTeamBToTeamC(), displayTeam(...), displayAllTeams(), displayTeamSizes().
3. **SortingOperations.java**
    - sortByHeight(), sortByWeight(), findMinMaxValues(), sortSingleTeam(...).
    - Semua menerapkan Collections.sort(...) dan/atau Java Stream untuk cari min/max.
4. **SearchOperations.java**
    - searchHeightInTeamB(), searchWeightInTeamA(), existsHeightInTeamA(int).
    - Helper searchSpecificHeight(...) dan searchSpecificWeight(...).
    - (Opsional) checkCommonValues(), searchCustomValue(...).

Main.java (entry point)

```
public class Main {
    public static void main(String[] args) {
        // Cetak judul program
        // 1. Inisialisasi dataManager, sortingOps, searchOps
        // 2. Tampilkan data awal → dataManager.displayAllTeams()

        // BAGIAN 1
        // 1a. sortingOps.sortByHeight()
        // 1b. sortingOps.sortByWeight()
        // 1c. sortingOps.findMinMaxValues()
        // Demonstrasi tambahan:
        //    dataManager.copyTeamBToTeamC(); dataManager.displayTeam("Tim C");
        //    sortingOps.sortSingleTeam("Tim A","height",false);
        //    dataManager.displayTeamSizes();

        // BAGIAN 2
        // 2a. Cetak jumlah pemain per tim: dataManager.getTeamA().size(), dataManager.getTeamB().size()
        // 2b. searchOps.searchHeightInTeamB()
        // 2c. searchOps.searchWeightInTeamA()
        // 2d. System.out.printf( "2d. … ada? %s%n", searchOps.existsHeightInTeamA(168) ? "Ada" : "Tidak ada" );
    }
}
```

5. 5.1. Urutan eksekusi di main()

Cetak baris:

=== PROGRAM MANAJEMEN TIM FUTSAL ===

Implementasi Collection Sorting dan Searching

==========

1. Buat objek:

   TeamDataManager dataManager = new TeamDataManager();

SortingOperations sortingOps = new SortingOperations(dataManager);

SearchOperations searchOps = new SearchOperations(dataManager);

2. **Bagian data awal**:

   System.out.println("\n1. MENAMPILKAN DATA AWAL");

dataManager.displayAllTeams();

3. **Pisahkan tanda batas**:

   System.out.println("\n" + "=".repeat(60));

System.out.println("BAGIAN 1: SORTING DAN OPERASI COLLECTION");

System.out.println("=".repeat(60));

4. **1a**:

   System.out.println("\n1a. SORTING BERDASARKAN TINGGI BADAN");

sortingOps.sortByHeight();

5. **1b**:

   System.out.println("\n1b. SORTING BERDASARKAN BERAT BADAN");

sortingOps.sortByWeight();

6. **1c**:

   System.out.println("\n1c. NILAI MIN DAN MAX TINGGI BADAN & BERAT BADAN");

sortingOps.findMinMaxValues();

7. **Demonstrasi fitur tambahan**:

   System.out.println("\n" + "=".repeat(60));

System.out.println("DEMONSTRASI FITUR TAMBAHAN");

System.out.println("=".repeat(60));

```
System.out.println("\nMenyalin Tim B ke Tim C...");
dataManager.copyTeamBToTeamC();
dataManager.displayTeam("Tim C");

System.out.println("\nContoh sorting Tim A berdasarkan tinggi (descending):");
sortingOps.sortSingleTeam("Tim A", "height", false);

dataManager.displayTeamSizes();
```

8. **Bagian 2 – Batas & Judul**:

```
System.out.println("\n" + "=".repeat(60));
System.out.println("BAGIAN 2: BINARY SEARCH PADA ARRAYLIST");
System.out.println("=".repeat(60));
```

9. **2a**:

```
System.out.println("2a. Jumlah pemain per tim:");
System.out.println("    Tim A: " + dataManager.getTeamA().size() + " pemain");
System.out.println("    Tim B: " + dataManager.getTeamB().size() + " pemain");
```

10. **2b**:

```
searchOps.searchHeightInTeamB();
```

11. **2c**:

```
searchOps.searchWeightInTeamA();
```

12. **2d**:

```
int cekHeight = 168;
boolean ada168 = searchOps.existsHeightInTeamA(cekHeight);
System.out.printf("\n2d. Apakah ada pemain di Tim A dengan tinggi %d cm? %s%n",
                  cekHeight, (ada168 ? "Ada" : "Tidak ada"));
```

6. Cara Menjalankan & Contoh Output

Asumsikan seluruh file berada di satu folder, misalnya Pertemuan12:

Pertemuan12/

├── Main.java

├── Player.java

├── TeamDataManager.java

├── SortingOperations.java

├── SearchOperations.java

├── HeightComparator.java

└── WeightComparator.java

6.1. Kompilasi

Buka terminal/command prompt di folder Pertemuan12 kemudian jalankan:

javac *.java

Jika tidak ada error, akan dihasilkan banyak file .class, satu per masing-masing .java.

6.2. Menjalankan

Setelah kompilasi sukses:

java Main

Contoh output lengkap (termasuk Bagian 1 dan Bagian 2)

=== PROGRAM MANAJEMEN TIM FUTSAL ===

Implementasi Collection Sorting dan Searching

==========

1. MENAMPILKAN DATA AWAL

=== DATA TIM FUTSAL ===

--- Tim A ---

Player 1 (Tim A): Height=168cm, Weight=50kg

Player 2 (Tim A): Height=170cm, Weight=60kg

Player 3 (Tim A): Height=165cm, Weight=56kg

Player 4 (Tim A): Height=168cm, Weight=55kg

Player 5 (Tim A): Height=172cm, Weight=60kg

Player 6 (Tim A): Height=170cm, Weight=70kg

Player 7 (Tim A): Height=169cm, Weight=66kg

Player 8 (Tim A): Height=165cm, Weight=56kg

Player 9 (Tim A): Height=171cm, Weight=72kg

Player 10 (Tim A): Height=166cm, Weight=56kg

--- Tim B ---

Player 1 (Tim B): Height=170cm, Weight=66kg

Player 2 (Tim B): Height=167cm, Weight=60kg

Player 3 (Tim B): Height=165cm, Weight=59kg

Player 4 (Tim B): Height=166cm, Weight=58kg

Player 5 (Tim B): Height=168cm, Weight=58kg

Player 6 (Tim B): Height=175cm, Weight=71kg

Player 7 (Tim B): Height=172cm, Weight=68kg

Player 8 (Tim B): Height=171cm, Weight=68kg

Player 9 (Tim B): Height=168cm, Weight=65kg

Player 10 (Tim B): Height=169cm, Weight=60kg

==========

BAGIAN 1: SORTING DAN OPERASI COLLECTION

==========

1a. SORTING BERDASARKAN TINGGI BADAN

=== SORTING BERDASARKAN TINGGI BADAN (Ascending) ===

Player 3 (Tim A): Height=165cm, Weight=56kg

Player 8 (Tim A): Height=165cm, Weight=56kg

Player 3 (Tim B): Height=165cm, Weight=59kg

Player 10 (Tim A): Height=166cm, Weight=56kg

Player 4 (Tim B): Height=166cm, Weight=58kg

Player 2 (Tim B): Height=167cm, Weight=60kg

Player 1 (Tim A): Height=168cm, Weight=50kg

Player 4 (Tim A): Height=168cm, Weight=55kg

Player 5 (Tim B): Height=168cm, Weight=58kg

Player 9 (Tim B): Height=168cm, Weight=65kg

Player 7 (Tim A): Height=169cm, Weight=66kg

Player 10 (Tim B): Height=169cm, Weight=60kg

Player 2 (Tim A): Height=170cm, Weight=60kg

Player 6 (Tim A): Height=170cm, Weight=70kg

Player 1 (Tim B): Height=170cm, Weight=66kg

Player 9 (Tim A): Height=171cm, Weight=72kg

Player 8 (Tim B): Height=171cm, Weight=68kg

Player 5 (Tim A): Height=172cm, Weight=60kg

Player 7 (Tim B): Height=172cm, Weight=68kg

Player 6 (Tim B): Height=175cm, Weight=71kg

=== SORTING BERDASARKAN TINGGI BADAN (Descending) ===

Player 6 (Tim B): Height=175cm, Weight=71kg

Player 5 (Tim A): Height=172cm, Weight=60kg

Player 7 (Tim B): Height=172cm, Weight=68kg

Player 9 (Tim A): Height=171cm, Weight=72kg

Player 8 (Tim B): Height=171cm, Weight=68kg

Player 2 (Tim A): Height=170cm, Weight=60kg

Player 6 (Tim A): Height=170cm, Weight=70kg

Player 1 (Tim B): Height=170cm, Weight=66kg

Player 7 (Tim A): Height=169cm, Weight=66kg

Player 10 (Tim B): Height=169cm, Weight=60kg

Player 1 (Tim A): Height=168cm, Weight=50kg

Player 4 (Tim A): Height=168cm, Weight=55kg

Player 5 (Tim B): Height=168cm, Weight=58kg

Player 9 (Tim B): Height=168cm, Weight=65kg

Player 2 (Tim B): Height=167cm, Weight=60kg

Player 10 (Tim A): Height=166cm, Weight=56kg

Player 4 (Tim B): Height=166cm, Weight=58kg

Player 3 (Tim A): Height=165cm, Weight=56kg

Player 8 (Tim A): Height=165cm, Weight=56kg

Player 3 (Tim B): Height=165cm, Weight=59kg

1b. SORTING BERDASARKAN BERAT BADAN

=== SORTING BERDASARKAN BERAT BADAN (Ascending) ===

Player 1 (Tim A): Height=168cm, Weight=50kg

Player 4 (Tim A): Height=168cm, Weight=55kg

Player 3 (Tim A): Height=165cm, Weight=56kg

Player 8 (Tim A): Height=165cm, Weight=56kg

Player 10 (Tim A): Height=166cm, Weight=56kg

Player 4 (Tim B): Height=166cm, Weight=58kg

Player 5 (Tim B): Height=168cm, Weight=58kg

Player 3 (Tim B): Height=165cm, Weight=59kg

Player 2 (Tim A): Height=170cm, Weight=60kg

Player 5 (Tim A): Height=172cm, Weight=60kg

Player 2 (Tim B): Height=167cm, Weight=60kg

Player 10 (Tim B): Height=169cm, Weight=60kg

Player 9 (Tim B): Height=168cm, Weight=65kg

Player 7 (Tim A): Height=169cm, Weight=66kg

Player 1 (Tim B): Height=170cm, Weight=66kg

Player 7 (Tim B): Height=172cm, Weight=68kg

Player 8 (Tim B): Height=171cm, Weight=68kg

Player 6 (Tim A): Height=170cm, Weight=70kg

Player 6 (Tim B): Height=175cm, Weight=71kg

Player 9 (Tim A): Height=171cm, Weight=72kg

=== SORTING BERDASARKAN BERAT BADAN (Descending) ===

Player 9 (Tim A): Height=171cm, Weight=72kg

Player 6 (Tim B): Height=175cm, Weight=71kg

Player 6 (Tim A): Height=170cm, Weight=70kg

Player 7 (Tim B): Height=172cm, Weight=68kg

Player 8 (Tim B): Height=171cm, Weight=68kg

Player 7 (Tim A): Height=169cm, Weight=66kg

Player 1 (Tim B): Height=170cm, Weight=66kg

Player 9 (Tim B): Height=168cm, Weight=65kg

Player 2 (Tim A): Height=170cm, Weight=60kg

Player 5 (Tim A): Height=172cm, Weight=60kg

Player 2 (Tim B): Height=167cm, Weight=60kg

Player 10 (Tim B): Height=169cm, Weight=60kg

Player 3 (Tim B): Height=165cm, Weight=59kg

Player 4 (Tim B): Height=166cm, Weight=58kg

Player 5 (Tim B): Height=168cm, Weight=58kg

Player 3 (Tim A): Height=165cm, Weight=56kg

Player 8 (Tim A): Height=165cm, Weight=56kg

Player 10 (Tim A): Height=166cm, Weight=56kg

Player 4 (Tim A): Height=168cm, Weight=55kg

Player 1 (Tim A): Height=168cm, Weight=50kg

1c. NILAI MIN DAN MAX TINGGI BADAN & BERAT BADAN

--- Tim A ---

Tinggi Badan:

Min: 165cm (Player 3)

Max: 172cm (Player 5)

Berat Badan:

Min: 50kg (Player 1)

Max: 72kg (Player 9)

--- Tim B ---

Tinggi Badan:

Min: 165cm (Player 3)

Max: 175cm (Player 6)

Berat Badan:

Min: 58kg (Player 4)

Max: 71kg (Player 6)

--- Gabungan Tim A & B ---

Tinggi Badan:

Min: 165cm (Player 3 - Tim A)

Max: 175cm (Player 6 - Tim B)

Berat Badan:

Min: 50kg (Player 1 - Tim A)

Max: 72kg (Player 9 - Tim A)

==========

DEMONSTRASI FITUR TAMBAHAN

==========

Menyalin Tim B ke Tim C...

--- Tim C ---

Player 1 (Tim C): Height=170cm, Weight=66kg

Player 2 (Tim C): Height=167cm, Weight=60kg

Player 3 (Tim C): Height=165cm, Weight=59kg

Player 4 (Tim C): Height=166cm, Weight=58kg

Player 5 (Tim C): Height=168cm, Weight=58kg

Player 6 (Tim C): Height=175cm, Weight=71kg

Player 7 (Tim C): Height=172cm, Weight=68kg

Player 8 (Tim C): Height=171cm, Weight=68kg

Player 9 (Tim C): Height=168cm, Weight=65kg

Player 10 (Tim C): Height=169cm, Weight=60kg

Contoh sorting Tim A berdasarkan tinggi (descending):

--- Sorting Tim A berdasarkan tinggi (Descending) ---

Player 5 (Tim A): Height=172cm, Weight=60kg

Player 9 (Tim A): Height=171cm, Weight=72kg

Player 2 (Tim A): Height=170cm, Weight=60kg

Player 6 (Tim A): Height=170cm, Weight=70kg

Player 7 (Tim A): Height=169cm, Weight=66kg

Player 1 (Tim A): Height=168cm, Weight=50kg

Player 4 (Tim A): Height=168cm, Weight=55kg

Player 10 (Tim A): Height=166cm, Weight=56kg

Player 3 (Tim A): Height=165cm, Weight=56kg

Player 8 (Tim A): Height=165cm, Weight=56kg

=== JUMLAH PEMAIN ===

Tim A: 10 pemain

Tim B: 10 pemain

Tim C: 10 pemain

==========

BAGIAN 2: BINARY SEARCH PADA ARRAYLIST

==========

2a. Jumlah pemain per tim:

Tim A: 10 pemain

Tim B: 10 pemain

=== BINARY SEARCH TINGGI BADAN TIM B ===

Tinggi badan Tim B (sorted): [165, 166, 167, 168, 168, 169, 170, 171, 172, 175]

Tinggi 168cm:

- Jumlah pemain: 2

- Binary search result: Found at index 3

- Penjelasan: Ditemukan di posisi indeks 3 dalam list yang sudah terurut

Tinggi 160cm:

- Jumlah pemain: 0

- Binary search result: Not found (-1)

- Penjelasan: Tidak ditemukan. Jika ingin disisipkan, posisinya akan berada di indeks 0

=== BINARY SEARCH BERAT BADAN TIM A ===

Berat badan Tim A (sorted): [50, 55, 56, 56, 56, 60, 60, 66, 70, 72]

Berat 56kg:

- Jumlah pemain: 3

- Binary search result: Found at index 2

- Penjelasan: Ditemukan di posisi indeks 2 dalam list yang sudah terurut

Berat 53kg:

- Jumlah pemain: 0

- Binary search result: Not found (-1)

- Penjelasan: Tidak ditemukan. Jika ingin disisipkan, posisinya akan berada di indeks 2

2d. Apakah ada pemain di Tim A dengan tinggi 168 cm? Ada

7. Ringkasan Poin‑Poin Utama

- **Model data (Player)**: menyimpan id, height, weight, teamName, dengan method copyToTeam(...) untuk menyalin pemain ke tim lain.

- **Comparator (HeightComparator & WeightComparator)**: membandingkan atribut height atau weight sehingga mudah mengurut (Collections.sort(...)).

- **TeamDataManager**:

    1. Membangun data statis untuk Tim A dan Tim B di constructor.

2. Menyediakan getter untuk teamA, teamB, dan (setelah copy) teamC.

3. getAllPlayers() mengembalikan gabungan teamA + teamB + teamC.

4. copyTeamBToTeamC() menyalin pemain Tim B ke Tim C.

5. displayTeam(...), displayAllTeams(), displayTeamSizes() untuk menampilkan ke console.

- **SortingOperations**:

    1. sortByHeight() dan sortByWeight(): mengurut seluruh pemain gabungan A + B + C secara ascending & descending.

    2. findMinMaxValues(): cari nilai minimum dan maksimum tinggi dan berat per tim (Tim A, Tim B), dan opsional gabungan A + B.

    3. sortSingleTeam(...): mengurut satu tim tertentu saja (A, B, atau C), berdasarkan field (“height” atau “weight”) secara ascending/descending.

- **SearchOperations**:

    1. searchHeightInTeamB():

        1. Konversi teamB → ArrayList<Integer> heightsB

        2. Sort ascending, print daftar

        3. searchSpecificHeight(heightsB, 168) → hitung frequency, binarySearch, print detail

        4. searchSpecificHeight(heightsB, 160) → sama

    2. searchWeightInTeamA():

        1. Konversi teamA → ArrayList<Integer> weightsA

        2. Sort ascending, print daftar

        3. searchSpecificWeight(weightsA, 56) → hitung frequency, binarySearch, print

        4. searchSpecificWeight(weightsA, 53) → sama

    3. existsHeightInTeamA(int): secara ringkas men‑sort ArrayList<Integer> heightsA dari Tim A, lalu Collections.binarySearch(heightsA, targetHeight) untuk mengembalikan true/false jika nilai tersebut ada di list.

    4. Opsional: checkCommonValues() (analisis nilai tinggi/berat yang sama antara Tim A dan Tim B), searchCustomValue(...) (binary search untuk nilai kustom pada tim

manapun).

- **Main.java**: mengeksekusi seluruh rangkaian di atas sesuai urutan tugas:

    1. Tampilkan data awal (Tim A dan Tim B).

    2. Bagian 1 (sorting gabungan, min/max, copy Tim B → Tim C, sorting per tim, display sizes).

    3. Bagian 2 (jumlah pemain per tim, searchHeightInTeamB, searchWeightInTeamA, existsHeightInTeamA).