HW2 MLB Profit

說明

檔案中xdemo是改進前 demo是改進後的

介紹

計算 12 支 MLB 球隊在季後賽中的最低和最高收益，根據不同系列賽的場次、票價、主場和客場的分潤比例進行計算。

程式分為三個主要部分：

- Team 類別：用來儲存單一球隊的基本資訊，如球場名稱、座位數、季後賽滿座率、世界大賽滿座率和排名等。
- PlayoffCalculator 類別：負責計算各個球隊在季後賽中可能的最低和最高收益。它包含多個方法，用於執行不同的計算和檢查。
- mlb_Profit 主程式：執行整個計算流程並顯示結果。

Code Review

命名規範

在 Java 中，類別名稱採用大駝峰命名法（PascalCase），這樣程式碼看起來更有條理。例如，MLBProfit 這個類別的命名就符合 Java 的規範。

```
/**
 * MLBProfit is the main class for calculating playoff profits.
 */
public class MLBProfit {
    // 主程式碼
}
```

加上註解

所有類別、方法和欄位都有 Javadoc 註解，讓程式更容易理解，維護起來也比較方便。增強程式的可讀性，還讓後續維護者更容易理解程式邏輯。例如，在 PlayoffCalculator 類別中，我在每個方法前都加上了註解說明

```
/**
 * Calculates the minimum profit for a team.
 * @param team The team object.
 * @return Minimum profit.
 */
private double calculateMinProfit(final Team team) {
```

```
    // 計算最小收益的邏輯
}
```

變數的不可變性

在能確定變數值不會被修改的情況下，使用 final 修飾詞聲明變數，使變數成為不可變的（immutable），以提高程式的安全性與可預測性。例如，以下程式片段中，所有變數都被設為 final

```
public void calculateProfit() {
    for (final Team team : teams) {
        final double minProfit = calculateMinProfit(team);
        final double maxProfit = calculateMaxProfit(team);
        // 輸出結果
    }
}
```

工具類的設計

在實用工具類（utility classes）中，使用私有構造函數以防止不必要的實例化。例如，在 MLBProfit 類別中，我將構造函數設為私有，這是一種常見的最佳實踐

```
private MLBProfit() {
    // 私有構造函數
}
```

Preventive Programming

資料輸入完整性檢查

在處理資料時，我加入了多層的資料驗證，以確保資料的完整性和範圍正確。例如，在從 JSON 文件中讀取球隊資料時，我增加了資料驗證邏輯，確保所有欄位存在且範圍合法

```
private static boolean isValidTeamData(final JSONObject obj) {
    try {
        if (!obj.has("team") || !obj.has("seating_capacity") ||
            !obj.has("playoff_full_rate") || !obj.has("world_series_full_rate"))
            return false;
        }
        final int seatingCapacity = obj.getInt("seating_capacity");
        if (seatingCapacity <= 0) return false;

        final double playoffRate = obj.getDouble("playoff_full_rate");
        final double wsRate = obj.getDouble("world_series_full_rate");
        return playoffRate >= 0 && playoffRate <= 1 && wsRate >= 0 && wsRate <=
    } catch (JSONException e) {
```

```
        return false;
    }
}
```

異常處理

在讀取文件和解析 JSON 的過程中，我加入了異常處理，防止因檔案不存在、格式不正確或資料缺失而導致程式崩潰。

```
public static List<Team> loadTeamsFromJson(final String filePath) {
    final List<Team> teams = new ArrayList<>();
    try {
        final String jsonData = new String(Files.readAllBytes(Paths.get(filePath
        final JSONArray jsonArray = new JSONArray(jsonData);
        // 資料解析
    } catch (IOException | JSONException e) {
        System.err.println("Error loading teams from JSON: " + e.getMessage());
    }
    return teams;
}
```

避免在迴圈中創建新物件

在迴圈中避免創建重複性物件。

```
final int[] games = {3, 5, 7, 7};
final double[] prices = {WC_PRICE, WC_PRICE, WC_PRICE, WS_PRICE};
final double[] rates = {team.getPlayoffRate(), team.getPlayoffRate(), team.getPl

for (int i = 0; i < games.length; i++) {
    final int homeGames = (i == 0) ? games[i] : (games[i] / 2 + 1);
    totalProfit += team.getSeatingCapacity() * rates[i] * prices[i] * HOME_SHARE
}
```

預防性判斷

在程式邏輯中加入邊界值檢查，確保輸入數據和計算結果不會超出預期範圍。

```
if (seatingCapacity <= 0 || playoffRate < 0 || playoffRate > 1 || wsRate < 0 ||
    // 無效資料處理
}
```