

HW3

3a 封殺

主程式

主程式有寫一個main，用測試檔只會測試methon裡面的程式而不會有測試到main。但用intellij裡面內建的涵蓋度測試時會把main的程式也算進去，所以測試時要先把main刪掉才行。

```
public List<String> getForceOutBases(String bases) {
    // 如果輸入是空的或是 "x"，代表沒有人壘，只有一壘可以封殺
    if (bases == null || bases.trim().isEmpty() || bases.trim().equals("x")) {
        return Collections.singletonList("1B");
    }

    // 解析目前壘包狀態
    Set<String> currentBases = new HashSet<>(
        Arrays.asList(bases.split(",\\s*"))
    );

    // 驗證輸入格式
    for (String base : currentBases) {
        if (!base.equals("1B") && !base.equals("2B") && !base.equals("3B")) {
            throw new IllegalArgumentException("無效的壘包標示: " + base + "。請使用 1B、2B 或 3B。");
        }
    }

    List<String> forceOutBases = new ArrayList<>();

    // 一壘永遠可以封殺 (因為打者會往一壘跑)
    forceOutBases.add("1B");

    // 如果一壘有人，二壘可以封殺
    if (currentBases.contains("1B")) {
        forceOutBases.add("2B");
    }

    // 如果一二壘都有人，三壘可以封殺
    if (currentBases.contains("1B") && currentBases.contains("2B")) {
        forceOutBases.add("3B");
    }

    // 如果一二三壘都有人，本壘可以封殺
    if (currentBases.contains("1B") &&
        currentBases.contains("2B") &&
        currentBases.contains("3B")) {
        forceOutBases.add("HB");
    }

    return forceOutBases;
}
```

測試程式

```

@BeforeEach
void setUp() {
    checker = new ForceOutChecker();
}

@Test
@DisplayName("當沒有人人在壘時，只有一壘可以封殺")
void testEmptyBases() {
    List<String> result = checker.getForceOutBases("x");
    assertEquals(Arrays.asList("1B"), result, "空壘時應該只有一壘可以封殺");
}

@Test
@DisplayName("當一壘有人時，一、二壘可以封殺")
void testRunnerOnFirst() {
    List<String> result = checker.getForceOutBases("1B");
    assertEquals(Arrays.asList("1B", "2B"), result, "一壘有人時應該可以在一、二壘形成封殺");
}

@Test
@DisplayName("當一、三壘有人時，一、二壘可以封殺")
void testRunnersOnFirstAndThird() {
    List<String> result = checker.getForceOutBases("1B,3B");
    assertEquals(Arrays.asList("1B", "2B"), result, "一三壘有人時應該可以在一、二壘形成封殺");
}

@Test
@DisplayName("當只有三壘有人時，只有一壘可以封殺")
void testRunnerOnThird() {
    List<String> result = checker.getForceOutBases("3B");
    assertEquals(Arrays.asList("1B"), result, "三壘有人時應該只有一壘可以封殺");
}

@Test
@DisplayName("當一、二壘有人時，一、二、三壘可以封殺")
void testRunnersOnFirstAndSecond() {
    List<String> result = checker.getForceOutBases("1B,2B");
    assertEquals(Arrays.asList("1B", "2B", "3B"), result, "一二壘有人時應該可以在一、二、三壘形成封殺");
}

@Test
@DisplayName("當二、三壘有人時，只有一壘可以封殺")
void testRunnersOnSecondAndThird() {
    List<String> result = checker.getForceOutBases("2B,3B");
    assertEquals(Arrays.asList("1B"), result, "二三壘有人時應該只有一壘可以封殺");
}

@Test
@DisplayName("當滿壘時，所有壘包都可以封殺")
void testBasesLoaded() {
    List<String> result = checker.getForceOutBases("1B,2B,3B");
    assertEquals(Arrays.asList("1B", "2B", "3B", "HB"), result, "滿壘時所有壘包都應該可以封殺");
}

@Test
@DisplayName("當輸入為null時，只有一壘可以封殺")
void testNullInput() {
    List<String> result = checker.getForceOutBases(null);
    assertEquals(Arrays.asList("1B"), result, "輸入為null時應該只有一壘可以封殺");
}



@Test
@DisplayName("當輸入為空字串時，只有一壘可以封殺")
void testEmptyString() {
    List<String> result = checker.getForceOutBases("");
    assertEquals(Arrays.asList("1B"), result, "輸入為空字串時應該只有一壘可以封殺");
}

@Test
@DisplayName("當輸入格式錯誤時，應拋出IllegalArgumentException")
void testInvalidInput() {
    assertThrows(IllegalArgumentException.class,
        () -> checker.getForceOutBases("1B,4B"),
        "無效的壘包標示應該拋出IllegalArgumentException");
}

@Test
@DisplayName("當二壘有人時，只有一壘可以封殺")
void testRunnerOnSecond() {
    List<String> result = checker.getForceOutBases("2B");
    assertEquals(Arrays.asList("1B"), result, "二壘有人時應該只有一壘可以封殺");
}

```

涵蓋度100%

| Element ^ | Class, % | Method, % | Line, % | Branch, % |
|--|------------|------------|--------------|---------------|
| ✓  org.example.baseball | 100% (1/1) | 100% (1/1) | 100% (18/18) | 100% (26/2... |
|  ForceOutChecker | 100% (1/1) | 100% (1/1) | 100% (18/18) | 100% (26/2... |

測試涵蓋度

1. 條件覆蓋（Condition Coverage）：

無人上壘：`testEmptyBases()`，測試當沒有跑者時的情況。

單一跑者情況：

- 一壘有人：`testRunnerOnFirst()`
- 二壘有人：`testRunnerOnSecond()`
- 三壘有人：`testRunnerOnThird()`

複合跑者情況：

- 一、二壘有人：`testRunnersOnFirstAndSecond()`
- 一、三壘有人：`testRunnersOnFirstAndThird()`
- 二、三壘有人：`testRunnersOnSecondAndThird()`
- 滿壘：`testBasesLoaded()`

2. 異常處理測試（Exception Handling Testing）：

空輸入：

- 輸入為 `null`：`testNullInput()`
- 輸入為空字串：`testEmptyString()`

無效輸入：

- 格式錯誤的輸入：`testInvalidInput()`，例如"1B,4B"

邊界測試

1. 下界測試：

無人上壘：`testEmptyBases()`，測試當沒有任何跑者時的情況，這是壘上跑者數量的最小值。

2. 上界測試：

滿壘： `testBasesLoaded()`，測試當所有壘包都有跑者時的情況，這是壘上跑者數量的最大值。

3. 單一壘包測試：

測試了每個壘包單獨有跑者的情況（第一、二、三壘），這些都是邊界條件，因為它們代表了跑者數量和位置的最小變化。

4. 異常邊界測試：

空值輸入： `testNullInput()` 和 `testEmptyString()`，測試方法在接收到空值或空字串時的反應。

無效值輸入： `testInvalidInput()`，測試方法在接收到無效壘包標示（如"4B"）時是否正確拋出 `IllegalArgumentException`。

3b測試hw1

主程式

```

class Team {
    String name, league, division;
    int wins, losses; // Seed or ranking based on performance

    public Team(String name, int wins, int losses, String league, String division) {
        this.name = name;
        this.wins = wins;
        this.losses = losses;
        this.league = league;
        this.division = division;
    }

    public String getName() {
        return name;
    }

    public int getWins() {
        return wins;
    }

    @Override
    public String toString() {
        //return String.format("%-4s Wins: %d Losses: %d league: %s division: %s", name, wins,
        // losses, league, division);
        return String.format("%-4s", name); // Ensure fixed width for team names
    }
}

public class Main {

    // Method to build and print the playoff bracket for American League
    public static void generateALBracket(List<Team> teams) {
        System.out.println("AMERICAN LEAGUE");
        printALBracket(teams);
        System.out.println();
    }

    // Method to build and print the playoff bracket for National League
    public static void generateNLBracket(List<Team> teams) {
        printNLBracket(teams);
        System.out.println("NATIONAL LEAGUE");
    }

    // Helper method to print the American League bracket
    private static void printALBracket(List<Team> teams) {
        // Preventive Programming: Ensure we have exactly 6 teams per league
        //assert teams.size() == 6; "Error: Each league should have at least 6 teams.";
        if (teams.size() < 6) {
            System.out.println("Error: Each league should have at least 6 teams.");
            return;
        }

        // Format playoff matches based on your desired output format
        System.out.println(teams.get(5) + "6 -----");
        System.out.println(teams.get(2) + "3 ----- 7 -----");
        System.out.println("    " + teams.get(1) + "2 ----- 7");
        System.out.println(teams.get(4) + "5 -----");
        System.out.println(teams.get(3) + "4 ----- 7 -----");
        System.out.println("    " + teams.get(0) + "1 ----- 7 ----- 7");
        System.out.println("    " + teams.get(0) + "1 ----- 7");
    }

    // Helper method to print the National League bracket
    private static void printNLBracket(List<Team> teams) {
        // Preventive Programming: Ensure we have exactly 6 teams per league
        //assert teams.size() == 6; "Error: Each league should have at least 6 teams.";
        if (teams.size() < 6) {
            System.out.println("Error: Each league should have at least 6 teams.");
            return;
        }

        // Format playoff matches for National League
        System.out.println(teams.get(5) + "6 ----- 7 ----- 7");
        System.out.println(teams.get(2) + "3 -----");
        System.out.println("    " + teams.get(1) + "2 -----");
        System.out.println(teams.get(4) + "5 ----- 7 ----- 7");
        System.out.println(teams.get(3) + "4 -----");
        System.out.println("    " + teams.get(0) + "1 -----");
    }

    // Method to read teams from a txt file and sort them
    public static Map<String, List<Team>> readAndSortTeamsFromFile(String filename) {
        Map<String, List<Team>> leagueMap = new LinkedHashMap<>();
        List<Team> ALTeams = new ArrayList<>();
        List<Team> NLTeams = new ArrayList<>();
        boolean isNL = false; // Flag to separate leagues

        try (BufferedReader br = new BufferedReader(new FileReader(filename))) {
            String line;
            while ((line = br.readLine()) != null) {
                line = line.trim();
                if (line.isEmpty()) {
                    // Empty line separates the leagues
                    isNL = true;
                    continue;
                }

                // Split the line into team name and seed
                String[] parts = line.split("\\s+");
                if (parts.length < 5) {
                    System.out.println("Skipping malformed line: " + line);
                    continue;
                }
                String teamName = parts[0];
                int wins = Integer.parseInt(parts[1]);
                int losses = Integer.parseInt(parts[2]);
                String league = parts[3];
                String division = parts[4];

                Team team = new Team(teamName, wins, losses, league, division);

                // Add teams to the appropriate league list
                if (isNL) {
                    NLTeams.add(team);
                } else {
                    ALTeams.add(team);
                }
            }

            // Sort teams based on seed (lower seed number is better)
            Collections.sort(ALTeams,
                Comparator.comparingInt(Team::getWins).reversed(), thenComparing(Team::getName));
            Collections.sort(NLTeams,
                Comparator.comparingInt(Team::getWins).reversed(), thenComparing(Team::getName));

        } catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage());
        }

        // Add both leagues' teams to the map
        leagueMap.put("American League", ALTeams);
        leagueMap.put("National League", NLTeams);

        return leagueMap;
    }

    public static void main(String[] args) {
        // Read and sort the teams from the txt file
        Map<String, List<Team>> leagueMap = readAndSortTeamsFromFile("src/teams.txt");

        // Generate and print the playoff bracket for both leagues
        generateALBracket(leagueMap.get("American League"));
        generateNLBracket(leagueMap.get("National League"));
    }
}

```

測試程式

```
class MainTest {
    @Test
    void readAndSortTeamsFromFile() {
        Map<String, List<Team>> leagueMap = Main.readAndSortTeamsFromFile("src/teams.txt");
        List<Team> alTeams = leagueMap.get("American League");
        assertEquals("AL wrong sorting",
            ()->assertEquals("BAL", alTeams.get(0).getName()),
            ()->assertEquals("TB", alTeams.get(1).getName()),
            ()->assertEquals("HOU", alTeams.get(2).getName()),
            ()->assertEquals("TEX", alTeams.get(3).getName()),
            ()->assertEquals("TOR", alTeams.get(4).getName()),
            ()->assertEquals("SEA", alTeams.get(5).getName()));

        List<Team> nlTeams = leagueMap.get("National League");
        assertEquals("NL wrong sorting",
            ()->assertEquals("ATL", nlTeams.get(0).getName()),
            ()->assertEquals("LAD", nlTeams.get(1).getName()),
            ()->assertEquals("MIL", nlTeams.get(2).getName()),
            ()->assertEquals("PHI", nlTeams.get(3).getName()),
            ()->assertEquals("AZ", nlTeams.get(4).getName()),
            ()->assertEquals("MIA", nlTeams.get(5).getName()));
    }

    @Test
    void main() {
        Map<String, List<Team>> leagueMap = Main.readAndSortTeamsFromFile("src/teams.txt");

        // 測試是否正確跳過錯誤行
        List<Team> alTeams = leagueMap.get("American League");
        assertNotNull(alTeams);
        assertTrue(alTeams.size() > 0);

        List<Team> nlTeams = leagueMap.get("National League");
        assertNotNull(nlTeams);
        assertTrue(nlTeams.size() > 0);
    }
}
```

1. `readAndSortTeamsFromFile` 測試方法

測試從檔案中正確讀取和排序球隊的功能。

讀取 `src/teams.txt` 中的球隊資料後，檢查：

- 是否正確分類為 **American League** 和 **National League**。
- 各聯盟中球隊排序是否正確。

測試條件：

- 使用 `assertAll` 檢查每一個聯盟中的球隊排序。

- 驗證排序結果是否與預期一致。

2. `main` 測試方法

測試主程式整體邏輯是否正確：

- 測試是否能正確從檔案中讀取資料。
 - 確認錯誤行是否被正確跳過。
 - 確認生成的球隊列表不為空。
-