Lab 3: Testing

Team Members:

1. Dhrumil Amish Shah

Date:

2021-01-28

—

Subject:

Software Development Concepts

—

Professor:

Matthew Amy

Part 1 – Understand the problem

Q1.) Clarify any ambiguities that your team may have about the problem. Record these clarifications.

Column (-) was unclear in the leaderboard. After spending 15 to 20 minutes on the example given in the PDF, it was clear that (-) column of each team shows the total points scored by the opposite team in all matches.

Example:

- Two reacordGameOutcome(team1, team2, scoreTeam1, scoreTeam2) are as below:
  Anchor, Salty, 15, 12
  Anchor, RC, 12, 7
- Leaderboard is:

Team	W	L	T	+	-
Anchor	2	0	0	27	19
Salty	0	1	0	12	15
RC	0	1	0	7	12

- Here,
  Anchor (-) column: 12(team2 - Salty score) + 7(team2 - RC score) = 19
  Salty (-) column: 15(team1 - Anchor) = 15
  RC (-) column: 12(team1 - Anchor) = 12

Q2.) Identify the boundary conditions that exist in the problem statement. Record these boundaries.

- Maximum league space is of 24 teams. The range is [0, 24].
- Team name length is according to standards defined.
  - Example: [3, 15], i.e., Minimum 3, and Maximum 15
- Team score is between 0 to max_score_allowed where max_score_allowed is the maximum value datatype can hold (Here int). The range is [0, max_score_allowed].
- For each team,
  - Total games played = won + loss + tie
- To prepare leaderboard, iterate through the number of teams in the league. The range is [0, number_of_teams].

Q3.) Outline the type of control flow that _any_ implementation for the problem will need to follow, based on the problem statement. Record these flows.

- **Path testing:** To ensure all paths are covered.
- **Structured basis testing:** To ensure each statement exercised at least once.
- **Expression testing:** To ensure program behaves correctly on the execution of different expressions.

Q4.) Define what would be perceived as the normal order in which methods would be invoked for the problem.

The regular order is as follow:

1.) Call **addTeam()** n times where n is the max number of teams allowed or less than that (i.e., 24 or less). After each call to **addTeam()**, **createLeaderBoard()** creates a leaderboard for the league with new team added every time. Since no match played, the leader board will be arranged based on team names lexicographically.
2.) After every match played, **recordGameOutcome()** and **createLeaderBoard()** are called consecutively. After every match, the leaderboard updates and will be ordered based on most games win. If a tie happens, then the order of tie-breaking, in order of precedence, is:
   - most games won
   - most games tied
   - most points scored by the team
   - highest difference of points scored to points lost
   - most games played
   - lexicographic order of the team names

Part 2 – Create test cases

Q1.) Identify a set of input validation tests for each method as well as the expected outcome for each case.

Method 1:

public boolean addTeam(String teamName)

- ⇨ Already 24 teams added.
- ⇨ teamName is null.
- ⇨ teamName is an empty string.
- ⇨ teamName is not according to a defined team name standard (Example: Contains a character like @ that is not allowed).
- ⇨ Length of teamName is not valid (Example: Must be greater than three and less than fifteen inclusive).
- ⇨ teamName already added.

Expected outcome for the above test cases is that team will not be added in the league. The method will return **false**.

Method 2:

public boolean recordGameOutcome(String team1, String team2, int scoreTeam1, int scoreTeam2)

- ⇨ team1 or team2 is null.
- ⇨ team1 or team2 is an empty string.
- ⇨ team1 or team2 is not according to a defined team name standard. (Example: Contains a character like @ that is not allowed).
- ⇨ Length of team1 or team2 is not valid (Example: Must be greater than three and less than fifteen inclusive).
- ⇨ team1 or team2 does not exist in the league.
- ⇨ scoreTeam1 or scoreTeam2 is less than 0 (i.e., Negative).
- ⇨ scoreTeam1 or scoreTeam2 is greater than the upper limit (If defined).

Expected outcome for the above test cases is that game will not be recorded. The method will return **false**.

Method 3:
public String createLeaderBoard( )

- ⇨ No team exists in the league (Expected outcome is null)
- ⇨ No matches played in the league (Expected outcome is teams arranged lexicographically)

Q2.) Identify a set of boundary tests for these boundary cases.

Method 1:
public boolean addTeam(String teamName)

- ⇨ League space is 24 teams maximum (So boundary cases are when we try to add 23rd and 24th team).
  - 23rd and 24th team is allowed but, the 25th team is not.
- ⇨ Length of teamName. (Example: Must be greater than three and less than fifteen).
  - Must be in the range [3, 15] but, cannot be 2 or less or 16 and more.

Method 2:
public boolean recordGameOutcome(String team1, String team2, int scoreTeam1, int scoreTeam2)

- ⇨ Length of team1 and team2. (Example: Must be greater than three and less than fifteen).
  - Must be in the range [3, 15] but, cannot be 2 or less or 16 and more.
- ⇨ scoreTeam1 and scoreTeam2 must be between 0 and the upper_limit_value (If any).
  - Must be in the range [0, upper_limit_value] but, cannot be 0 or less (i.e., Negative value) or (upper_limit_value + 1) or more.
- ⇨ The number of games won, lost, and tied for each team has to be between 0 and the number of games played inclusively. (i.e. [0, number_of_games_played]).
  - Also, won + lost + tied = number of games played

Method 3:
public String createLeaderBoard( )

- ⇨ When no teams exist in the league and createLeaderBoard() is called.
- ⇨ Iterate through the number of teams in the league to prepare leaderboard. Allowed range is [0, number_of_teams].

Q3.) Identify a set of control flow tests.

Test case for method 1:
public boolean addTeam(String teamName)

- ⇨ (code block 1) Already 24 teams added, return false else go to (code block 2).
- ⇨ (code block 2) teamName is null, return false else go to (code block 3).
- ⇨ (code block 3) teamName is empty, return false else go to (code block 4).
- ⇨ (code block 4) teamName not according to the defined standard, return false else go to (code block 5).
- ⇨ (code block 5) teamName length is not valid, return false else go to (code block 6).
- ⇨ (code block 6) teamName added already, return false else go to (code block 7).
- ⇨ (code block 7) teamName is valid, add it to the league, return true.

Test case for method 2:
public boolean recordGameOutcome(String team1, String team2, int scoreTeam1, int scoreTeam2)

- ⇨ (code block 1) If team1 is null, return false else go to (code block 2).
- ⇨ (code block 2) If team2 is null, return false else go to (code block 3).
- ⇨ (code block 3) team1 is empty, return false else go to (code block 4).
- ⇨ (code block 4) team2 is empty, return false else go to (code block 5).
- ⇨ (code block 5) team1 not in the league, return false else go to (code block 6).
- ⇨ (code block 6) team2 not in the league, return false else go to
- ⇨ block 7).
- ⇨ (code block 7) team1 score not valid, return false else go to (code block 8).
- ⇨ (code block 7) team2 score not valid, return false else go to (code block 8).
- ⇨ (code block 5) Add team1, team2, scoreTeam1 and scoreTeam2 in table, return true.

Test case for method 3:
public String createLeaderBoard( )

- ⇨ (code block 1) No team exists, return null else go to (code block 2).
- ⇨ (code block 2) No match played, return teams arranged lexicographically else go to (code block 3).
- ⇨ (code block 3) Group matches by team name and arrange them with the most winning team first and if a tie happens, then arrange in order of precedence discussed in Part 1 - Q4 and return this string.

This control flow execution ensures structured basis, path, and expression testing.

Q4.) Identify a set of data flow tests based on your "normal" order.

- ⇨ 1 - addTeam(), 2 - recordGameOutcome(), 3 - createLeaderBoard()

Data flow tests

Test 1:Flow to cover leaderboard creation

- createLeaderBoard() // Prints nothing
- addTeam(t1)
- createLeaderBoard() // Leaderboard with 1 team
- addTeam(t2)
- createLeaderBoard() // Leaderboard with 2 teams arranged lexicographically
- addTeam(t3)
- createLeaderBoard() // Leaderboard with 2 teams arranged lexicographically
- recordGameOutcome(t1, t2, s1, s2) // Match recorded
- createLeaderBoard() // Leaderboard prepared
- recordGameOutcome(t1, t3, s3, s4) // Match recorded
- createLeaderBoard() // Leaderboard prepared

Each method can internally have a data flow test to check how conditions are executed.
Similar to Q3 but is focused on how variables change.

Q.) How complete you think that each set of tests are for the problem.

100%. Each method covers all the test cases. Input validation, control flow and data flow are all covered.

Questions

Q.) How, if at all, would input validation change if you were getting input from a user interface rather than as parameters to methods?

Input validation will not change even if we accept input from the user interface. We still need to check for all the test cases that we do for method parameters. Also, we need to ensure the data type of internal variables is same as that of variables we use to design user interface.

Q.) Explain whether or not boundary cases exist beyond checking the incoming input values. How does the idea of "control flow" change between black box tests and white box tests?

Boundary cases do exist beyond incoming input values like

- invalid input checks like -1 for score value

In Black Box testing, we are unaware of the internal structure and implementation. We are not safe to make any assumption on how input is processed.
In White Box testing, we are aware of the internal structure and implementation. We already know in advance on how input is processed.
White Box focuses on code structure, paths, conditions, and branches while Black Box focuses on end-user perspective.

Q.) Should all permutations of methods result in data flow tests? Explain.

At some extent, yes.
Reason:

- ⇨ Permutations of methods check how data is processed internally by methods and how the program is executed. Data flow test ensures that changes in variables do not result in unexpected results.
- ⇨ The idea behind this is to ensure that there are no unused variables, free variables after use, no uninitialized variables are used, and program correctly processes the data.