IMPORTANT INSTRUCTIONS

1. Download the file **CBT.zip**. It contains the following four files:
   (a) CBT_PART-1-2.pdf
   (b) CBT_PART_1_QP.java
   (c) inputCricketData.csv
   (d) CBT_PART_2_QP.java
2. Start eclipse and **create a new project** (named **CBT**). Copy and paste **CBT_PART_1_QP.java** and **CBT_PART_2_QP.java** files in the default package of your project. Rename the **CBT_PART_1_QP.java** with your **student ID number** (for example **P2022A7PSXXXX_P1.java**) and **CBT_PART_2_QP.java** as **P2022A7PSXXXX_P2.java**.
3. Partial (template) code is given to you. You are allowed to modify the portion of the code that is intentionally left blank with the comments
   /***** WRITE CODE FOR THIS METHOD *********/**
4. Periodically keep saving your work by pressing **Ctrl + Shift + S** (saves all your modifications). Once you finish writing the code or when the exam time gets over save all your work. Raise hand and call the invigilator.

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI
FIRST SEMESTER 2024-25
COMPUTER BASED TEST (OPEN BOOK)
OBJECT ORIENTED PROGRAMMING (CS F213), PART-1
TIME: 75 Min **MAX MARKS:** 35 **DATE:** 17/11/2024

You are required to implement functionalities for a Cricket Analytics application. The provided template code for this application contains multiple classes and methods related to cricket players, their roles, team information, and various data-handling operations. Your tasks involve implementing several methods, each responsible for performing specific actions like reading data from a file, writing data back, updating player statistics, and filtering data. You have been provided with the following files for this question:

1. CBT_PART_1_QP.java [Code template file where you need to write code for PART-1]
2. inputCricketData.csv [Input file which is required to be read as part of this question]

Instructions:

1. Before starting to write the code, read the entire question and go through the entire question. This is a very simple question. Just take a deep breath and write code peacefully!
2. **Do not modify any parts of the code not mentioned in the questions.**
3. **Do not change the names and signatures of the methods.**
4. **Do not add any member variables in any class.**

Tasks and Methods to be Implemented:

1. RunsComparator: compare Method [2 marks]

Write code for comparing runs scored by two players in descending order.
Return a negative value if the first player has more runs, a positive value if the second player has more runs, or zero if they have the same number of runs.

2. CricketDataHandler: readPlayersFromFile Method [9 marks]

Write code for reading player data from the input CSV file and creating a list of **Player** objects.

- Step 1: Create an empty list to store player details. [1 mark]
- Step 2: Open the specified file for reading data. [1 mark]
- Step 3: Ignore the first line since it contains the column names. [1 mark]
- Step 4: Read each line one by one until reaching the end of the file. [1 mark]
- Step 5: Split the line into different pieces of information. [1 mark]
- Step 6: Create a new player using this information. [1 mark]
- Step 7: Add the new player to the list. [1 mark]
- Step 8: Close the file after reading all data. [1 mark]
- Step 9: Return the complete list of players. [1 mark]

3. CricketDataHandler: writePlayersToFile Method [4 marks]

Write code to write the updated list of players back to the output CSV file. The format of the output file should be the same as that of the input file.

- Step 1: Prepare to write data into the specified file. [1 mark]
- Step 2: Write the column names as the first line of the file. [1 mark]
- Step 3: For each player in the list, convert their details to the desired format. [1 mark]

- Step 4: Write each player's information to the file. [1 mark]

4. CricketDataHandler: updatePlayerStats Method [5 marks]

Implement the method to update a player's stats (runs and wickets).

- Step 1: Go through each player in the list. [1 mark]
- Step 2: Check if the current player's name matches the given name. [1 mark]
- Step 3: If it matches, update the player's runs with the new value. Updated value will be the sum of the old runs and the argument runs. For example, if a player had 100 runs and the runs argument (to this method) is 50, their new total should be 150 runs. [1 mark]
- Step 4: Similarly, update the player's wickets with the new value. Updated value will be the sum of the old wickets and the argument wickets. For example, if they had 10 wickets and the wickets argument (to this method) is 2, their new total should be 12 wickets[1 mark]
- Step 5: If no player matches the given name, throw an IllegalArgumentException. [1 mark]

5. CricketDataHandler: calculateTeamAverageRuns Method [5 marks]

Write code to calculate the average runs scored by players of a specific team.

- Step 1: Filter players belonging to the specified team. [2 marks]
- Step 2: If no players for this team exist, throw an IllegalArgumentException exception. [1 mark]
- Step 3: Calculate the total runs scored by all players from this team. [1 mark]
- Step 4: Compute and return the average runs scored. [1 mark]

6. TeamFilter: filter Method [5 marks]

Write code to filter players by their team.

- Step 1: Create an empty list for players matching the criteria. [1 mark]
- Step 2: Go through each player in the players list. [1 mark]
- Step 3: If the player's team matches the given name, add them to the list. [2 marks]
- Step 4: Return the list containing all matching players. [1 mark]

7. AllRounderStatsFilter: filter Method [5 marks]

Write code to filter all-rounder players which satisfy the provided criteria (i.e. filter those all-rounders who have runs and wickets greater than or equal to the runs and wickets specified in the criteria respectively).

- Step 1: Create an empty list for players matching the criteria. [1 mark]
- Step 2: Go through each player in the list. [1 mark]
- Step 3: If the player is an all-rounder and meets the given criteria for both runs and wickets, add them to the list. [2 marks]
- Step 4: Return the list containing all matching players. [1 mark]

******* BEST OF LUCK *******

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI
FIRST SEMESTER 2024-25
COMPUTER BASED TEST (OPEN BOOK)
OBJECT ORIENTED PROGRAMMING (CS F213), PART-2

TIME: 45 Min **MAX MARKS:** 40 **DATE:** 17/11/2024

I. Game Description:

In PART-2, you will implement a classic card game "21" by implementing it as a multithreaded application. In this game players aim to reach a score as close as possible to 21 without exceeding it, while competing against each other. This game involves one dealer and exactly three players. The dealer does not play but manages the game and announces the final result. Each player is represented by a separate thread, and each turn and game action is synchronized to prevent conflicts. Here's how the game works:

1. **Initial Setup**:
   - ✓ The dealer thread starts the game by dealing two "cards" to each player thread. Each player receives one hidden card, known only to that player, and one visible card, which is displayed publicly (as a total visible score).
   - ✓ Cards are randomly generated integers between 1 and 10, inclusive, with each value equally likely. These values represent the points in this simplified version, so there are no suits or special cards.
2. **Game Play**:
   - ✓ After receiving initial cards, players take turns to request additional cards, each aiming to get as close to 21 as possible without going over. On each turn, a player may either:
     - o **Take a Card**: The player requests an additional card and adds its value to their total score.
     - o **Pass**: The player opts not to take more cards, effectively "locking in" their score. Once a player "PASSes", they cannot request additional cards.
   - ✓ A player who has passed is skipped in subsequent rounds.
   - ✓ The game progresses in rounds, with each round allowing each active player to take a turn until all players have passed. Once all players have passed, the game ends.
3. **Endgame and Scoring**:
   - ✓ After all players have passed, the dealer thread calculates and announces the final scores.
   - ✓ The winner is the player whose score is closest to 21 without exceeding it. In case of a tie or if all players exceed 21, no one wins.
4. **Output Requirements**:
   - ✓ Output must clearly indicate the sequence of actions, such as initial card distribution, each player's turn, round progression, and endgame results.
   - ✓ Display each player's decision, the dealer's announcements, and the final scores.

II. Example Output:

An example output for the game is provided to illustrate the required output format. Your output does not need to be identical but should clearly reflect each action and round of the game in a well-organized sequence.

```
P1 takes 10 (hidden)
P2 takes 1 (hidden)
P3 takes 3 (hidden)
P1 takes 1 (visible)
P2 takes 10 (visible)
```

```
P3 takes 10 (visible)
P1's turn. Current score: 11. (1) Take a card or (2) Pass? :1
P1 takes 5
P3's turn. Current score: 13. (1) Take a card or (2) Pass? :1
P3 takes 6
P2's turn. Current score: 11. (1) Take a card or (2) Pass? :1
P2 takes 9
P3's turn. Current score: 19. (1) Take a card or (2) Pass? :2
P3 passes.
P1's turn. Current score: 16. (1) Take a card or (2) Pass? :2
P1 passes.
P2's turn. Current score: 20. (1) Take a card or (2) Pass? :2
P2 passes.
P1 has 16
P2 has 20
P3 has 19
P2 wins with 20!
```

******* BEST OF LUCK *******