PROGRAMMING LOGIC AND DESIGN

PRLD5121

GROUP 2

Sbongkwanda Simelane
STUDENT NUMBER: ST10376730

Question 1

Q.1.1 Two programming models or paradigms that could be considered for the development of the application for sports administrators are:

1. **Procedural Programming Paradigm:**

The goal of procedural programming is to create a step-by-step method of instructions to solve a problem. It takes a top-down approach, breaking the problem down into smaller parts and developing procedures or functions to carry out each task. The procedural programming paradigm can aid in the organisation of the different duties and operations required in managing sports teams in the context of the sports administration application. Procedures can be established, for example, to add new players to teams, update player statistics, schedule matches, and generate reports.

2**. Object-Oriented Programming (OOP) Paradigm:**

Object-oriented programming revolves around the concept of objects, which are real-world entities that include properties (attributes) and behaviours (methods). This paradigm encourages code reusability, modularity, and ease of maintenance. In the instance of the sports administration application, OOP can be used to describe various entities involved as objects with their own qualities and behaviours, such as players, teams, matches, and administrators. A Player class, for example, can have characteristics such as name, age, and position, as well as methods for updating player information, calculating performance statistics, and generating player profiles.

Both paradigms have benefits that can be combined to obtain the necessary functionality. Procedural programming can aid in expressing and manipulating data in a more organised and modular manner, but object-oriented programming can help in representing and manipulating data in a more structured and modular manner.

Finally, the choice between the two paradigms will be determined by criteria such as the complexity of the application, the team's expertise with the paradigms, the necessity for code reusability, and future scalability needs.

Q.1.2 I can provide some guidance on ensuring the functionality of an application before delivering it to the client. Here are some steps you can take to increase the chances that the application will function as intended:

1. **Define clear requirements:** Begin by understanding the client's needs and establishing application requirements. This will be used as a guideline throughout the development process.

2. **Plan and design**: Before beginning development, create a clear plan and design for the application. Its architecture, user interface, and major functionalities must all be defined. This phase allows you to spot potential problems early on.

3. **Break down the development process:** Break down the development process into smaller, more manageable tasks or modules. This method allows you to address any issues sequentially and properly test each component.

4. **Conduct thorough testing:** To analyse the application at various levels, implement a thorough testing strategy that encompasses unit testing, integration testing, and system testing. To find flaws and guarantee adequate error handling, test both expected functionality and edge cases.

By following these procedures, you may improve the predictability and robustness of the application, reducing the likelihood of unexpected difficulties and guaranteeing that it performs properly when delivered to the client. (Farrell, 2018)

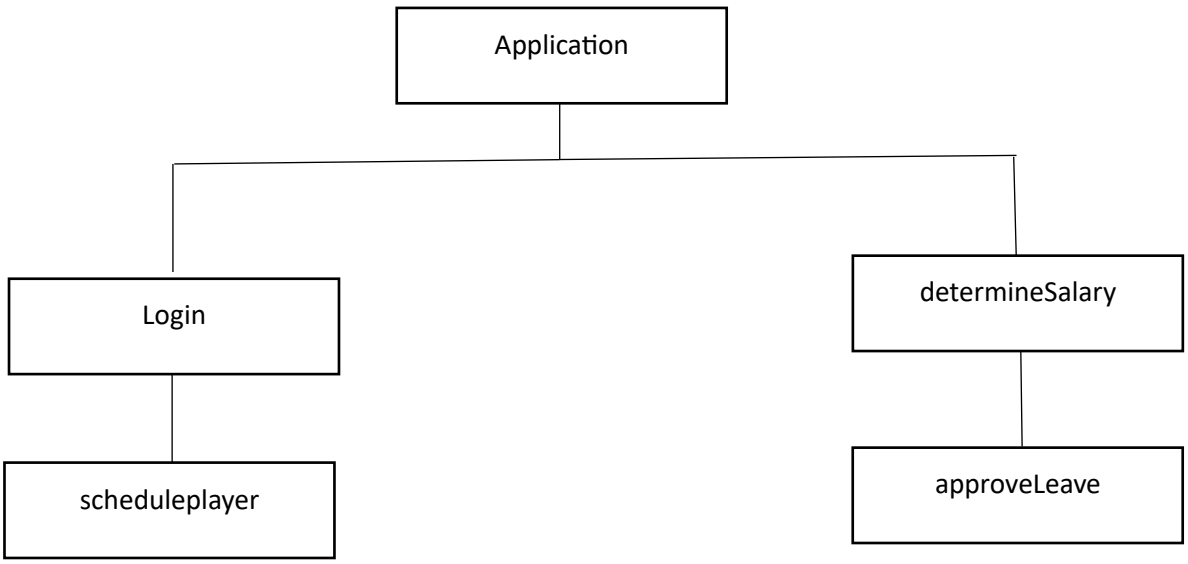
Q.1.3 Flowchart

Q.1.4.1. leave_days_applied = 0

Q.1.4.2. Salary = R75 000* Matches

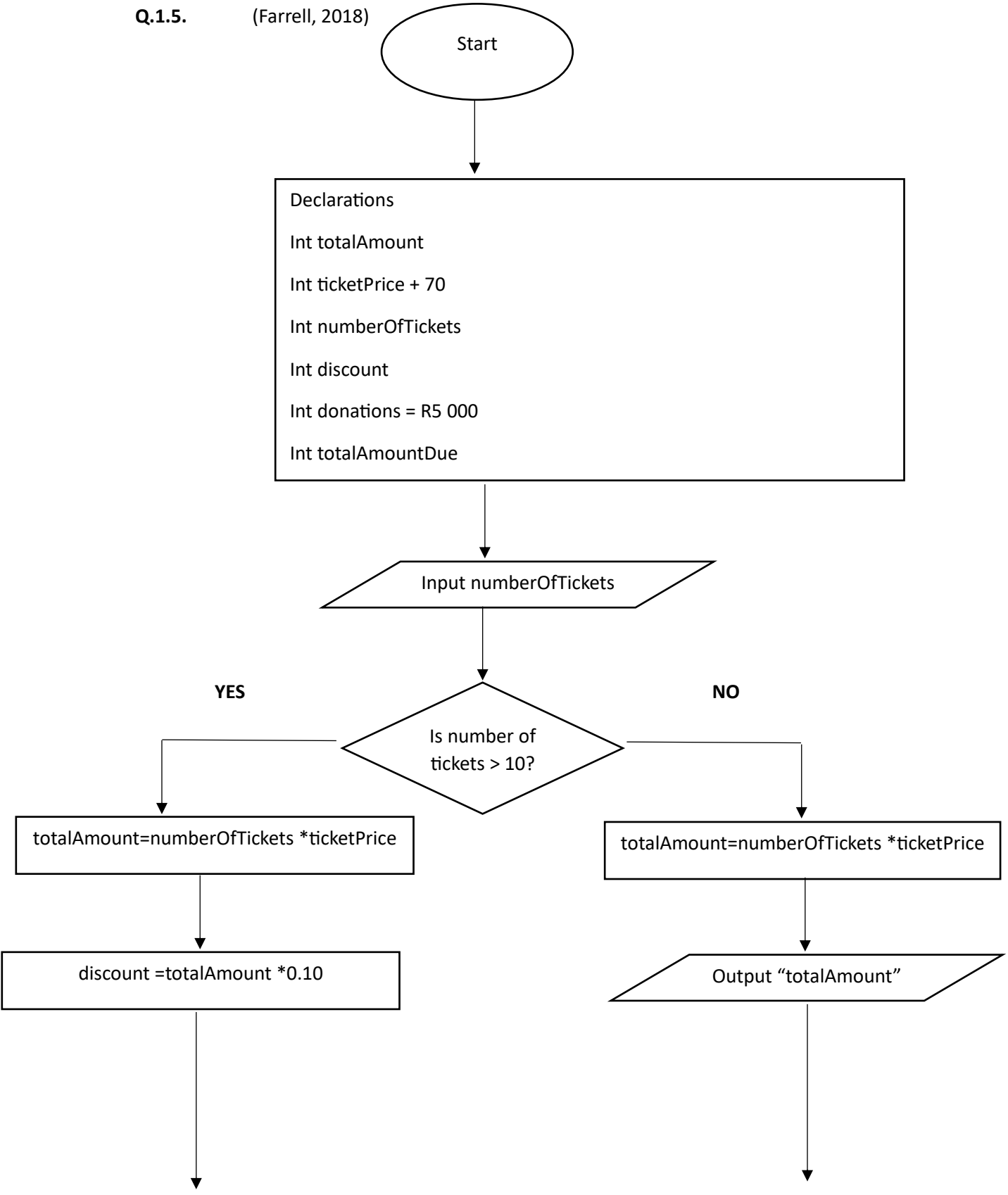
The variable “Matches” represents the number of matches played in each month. Multiplying this number by the standard rate of 75 000 per game will give you the salary of that player for that month.

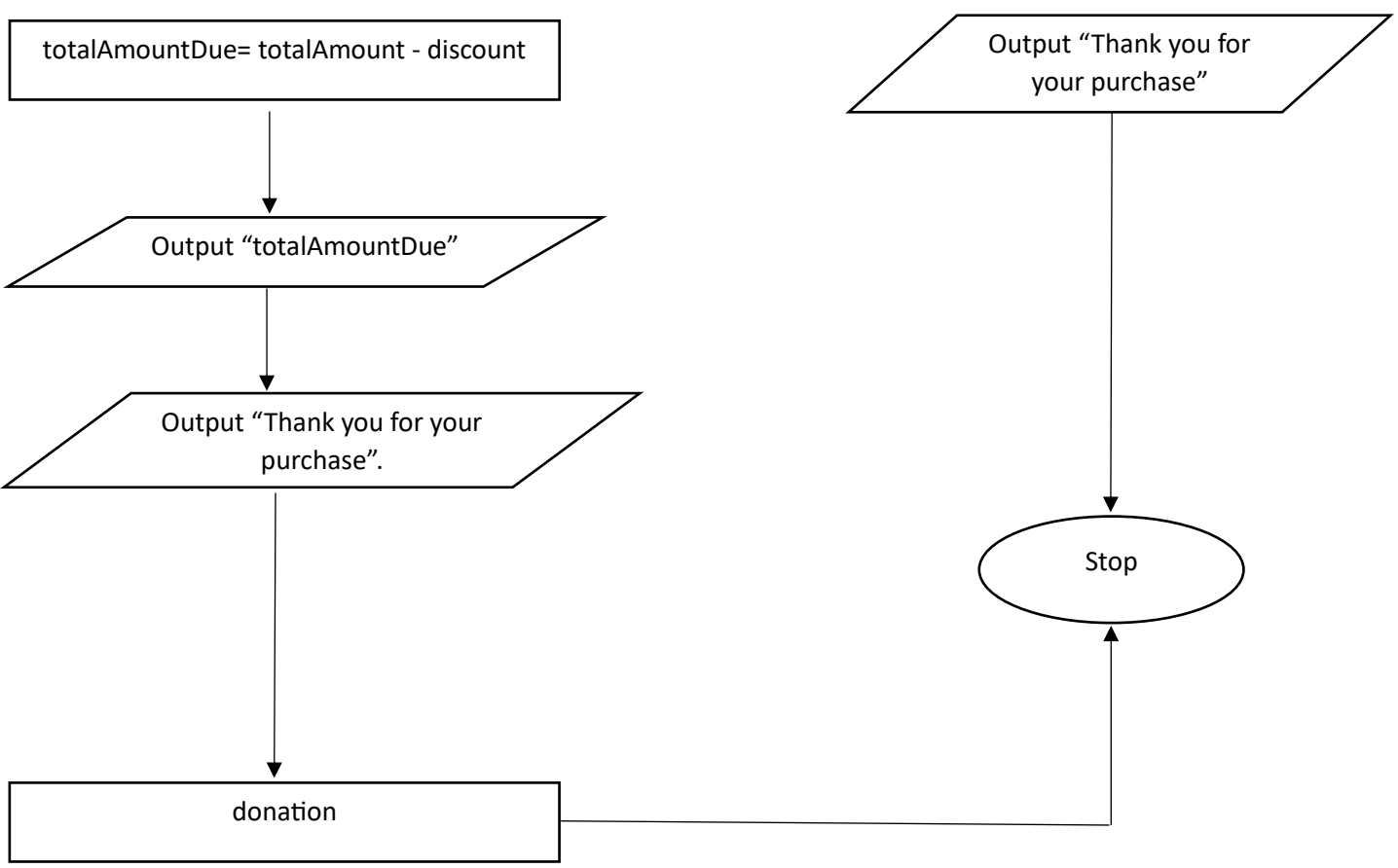
Q.1.4.3. (Farrell, 2018)



Q.1.4.4. // Calculate the player's salary.

Q.1.5. (Farrell, 2018)





Question 2

Q.2.1

Start

Declarations

Set bonusThreshold = 10 // Number of games required for bonus eligibility

Set bonusAmount = 5000 // Bonus amount for eligible players

Set basicRate = 24000 // Basic rate per game for junior players

Set gameCount = 0 // Initialize the game count to 0

Set isBonusEligible = false // Initialize bonus eligibility to false

// Loop to allow staff member to enter game status

Loop until gameCount >= bonusThreshold

Output “ Enter 'Yes' or 'No' “

```
                If      user entered 'Yes' then
    Increment gameCount by 1    // Increment game count if player played a    game
                end If


                        // Check if player is eligible for a bonus
                If          gameCount >= bonusThreshold then
                            Set isBonusEligible = true    // Player is eligible for a bonus
                end If


                        // Calculate player's salary
                Set salary = gameCount * basicRate // Multiply game count by basic rate to calculate
salary


                    // Display player's salary and bonus eligibility
                                    Display "Player's Salary: " + salary
                        If              isBonusEligible is true then
                                        Display "Bonus Eligible: Yes"
                                        Display "Bonus Amount: " + bonusAmount
                        else
                                        Display "Bonus Eligible: No"
                        endIf

Stop
```

Q.2.2

```
Start
Declarations
int playerAge
int  playerSalary
int premiumPercentage

// Prompt user for player's age and salary
Output “Enter player's age”
Input Enter player's age
                                        Output “player's salary”
 Input Enter player's salary

                                    // Check if player is older than 25 years
If        playerAge > 25 then
   // Perform risk assessment for high-risk, medium-risk, and low-risk individuals
          Output “Enter 'High', 'Medium', or 'Low' ”
           Input Risk assessment
 If        user entered 'High' then
          Set premiumPercentage = 7   // High-risk premium percentage
else if     user entered 'Medium' then
           Set premiumPercentage = 4   // Medium-risk premium percentage
 else if     user entered 'Low' then
           Set premiumPercentage = 2  // Low-risk premium percentage
 else
          Output  "Invalid risk assessment entered"
      // Exit the application or handle the error appropriately
end If
       // Calculate the premium based on the percentage of the salary
        Set premium = (premiumPercentage / 100) * playerSalary
```

```
        // Display the premium percentage
          Ouput "Premium Percentage: " + premiumPercentage + "%"

         // Display the premium amount
       Output  "Premium Amount: " + premium
else
       // Player is younger than or equal to 25 years, no risk assessment needed
      Output "No risk assessment needed. Player is under 25 years."
end If

Stop
```

Question 3

Q.3.1

```
Start

// Declarations
Set months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun"]   // Array to store month names
Set points = [30, 10, 5, 20, 25, 30]         // Array to store points earned

Set totalPoints = 0          // Variable to store total points
Set averagePoints = 0        // Variable to store average points
Set minPoints = points[0]      // Variable to store minimum points (initialize with the first element)
Set maxPoints = points[0]     // Variable to store maximum points (initialize with the first element)
Set minMonth = ""       // Variable to store the month with the minimum points
Set maxMonth = ""     // Variable to store the month with the maximum points

// Calculate total points and find the month with minimum and maximum points
for                    i = 0 to length of points - 1
                      Increment totalPoints by points[i]
```

```
    If                  points[i] < minPoints then
                       Set minPoints = points[i]
                      Set minMonth = months[i]
    end If

    If                  points[i] > maxPoints then
                       Set maxPoints = points[i]
                           Set maxMonth = months[i]
    end If
endfor

                        // Calculate average points per month
                    Set averagePoints = totalPoints / length of points

                     // Display the report
                  Display "Total Points: " + totalPoints
                 Display "Average Points per Month: " + averagePoints
                Display "Month with Lowest Points: " + minMonth
                Display "Month with Highest Points: " + maxMonth


Stop
```

Question 4

Q.4.1.

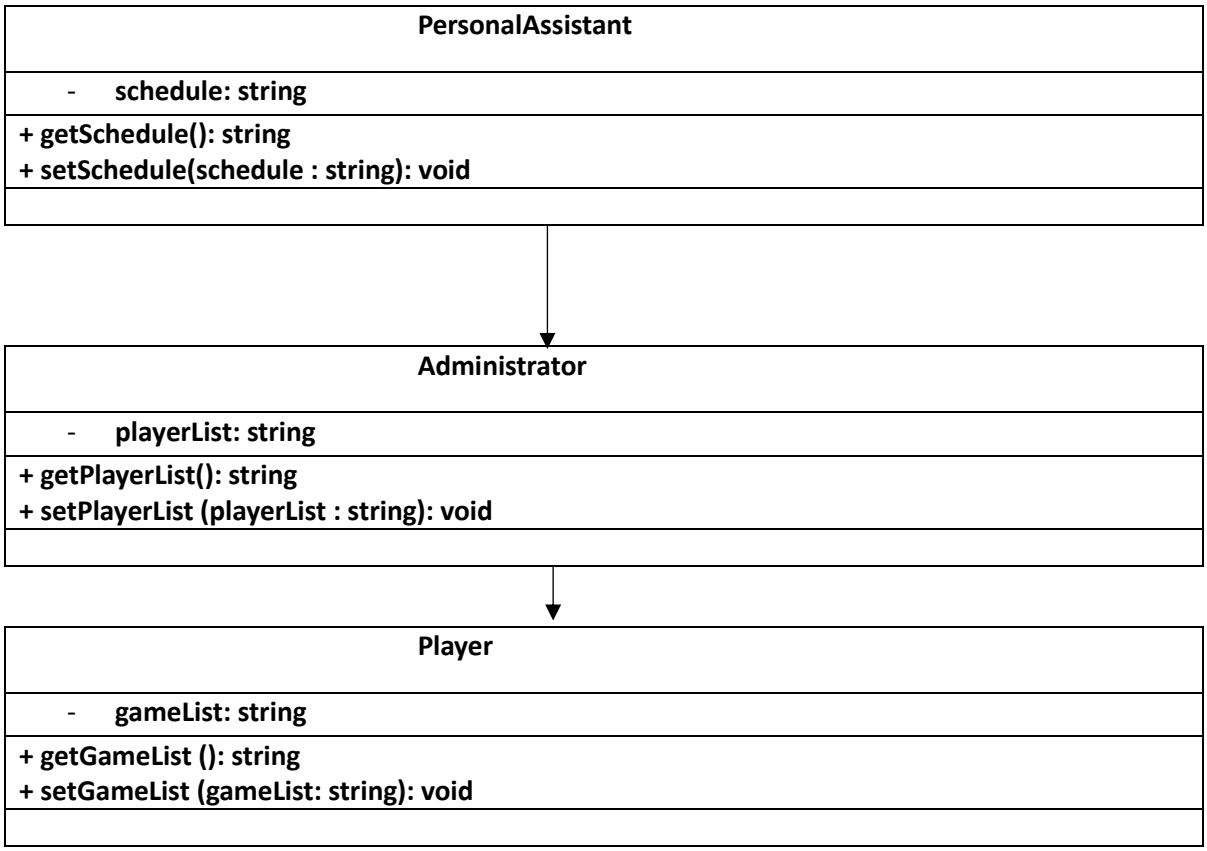
```
        Start
 Declarations
Set gameSalary = 70 000      // Salary earned per game
Set taxRate = 0.25          // Tax rate for higher tax bracket

// Function to calculate net salary
Function calculateNetSalary(gamesPlayed)
   Set grossSalary = gamesPlayed * gameSalary
   Set taxAmount = grossSalary * taxRate
   Set netSalary = grossSalary - taxAmount
   Return netSalary
End Function

Output "Enter the number of games played by the player:"
Input gamesPlayed
Set playerNetSalary = calculateNetSalary(gamesPlayed)
Display "Player Net Salary: R" + playerNetSalary

Stop
```

Q.4.2

The application (Overload Methods) can provide flexibility in handling different player statistics and provide appropriate performance measures based on the specific sport by employing overloaded methods. This method allows the logic and calculations to be encapsulated within relevant methods, making the code more modular and manageable. (Farrell, 2018)

Question 5

Q.5.1



Q.5.2.

Exception handling in the Sports Administration programme is critical for error detection, user-friendly error messages, application stability, quick debugging, and better security. It results in a more dependable, user-friendly, and maintainable software system.

REFERENCE

Farrell, J. (2018). *Programming Logic and Design.* Boston: Comprehensive.