Goals & Objectives

The goal of this program is to accept user input for basketball players for name, age, and height and return the player’s information whose age is less than the average age of all the players. The program should use three classes for storing and processing the basketball player information.

Functional Requirements

1. Prompt user for player Name, Age, and Height [in feet and inches] until input is blank.
2. Store each player in ArrayList for player class.
3. Sum the ages of all players and calculate the average age.
4. Iterate through players and find the oldest age less than average age.
5. Iterate through players and find the tallest height.
6. Output the Name, Age, and Height of the found player.

Pseudocode

Import Scanner Utility

Import ArrayList Utility

Function Main {

Declare players as new ArrayList;

Declare Double sumOfAges as 0

Declare Double averageAge as 0

Declare Integer tallestPlayerInches as 0

Declare Integer tallestPlayerIndex as 0

Declare Integer count as 0

Declare inputString as new Scanner

WHILE (true)

Output “Enter the player’s Name, Age, and Height (in feet and inches): “

Declare String line as inputString.nextLine

IF ( line.length == 0)

Break

ELSE

Add line to players as new Player

Close Input

FOR EACH player in players

Add player.getPlayerAge to sumOfAges

Calculate averageAge as sumOfAges / players.size

Output “The average age of all players is “ + averageAge

FOR EACH player in players

IF ( player.getPlayerAge < averageAge)

IF (player.getPlayerHeightInches > tallestPlayerInches)

tallestPlayerIndex = count

tallestPlayerInches = player.getPlayerHeightInches

Increment count

Output “The tallest player whose age is less than the average age is “ + players.get(tallestPlayerIndex).toString()

End

Class Player {

Declare playerName as private String

Declare playerAge as private Integer

Declare playerHeight as private Height

Constructor Player {}

Constructor Player (requires String s) {

Declare String List player as s.split(“ “)

Assign this.playerName as player[0]

Assign this.playerAge as Integer.parseInt(player[1])

Assign this.playerHeight as new Height(Integer.parseInt(player[2]), Integer.parseInt(player[3]))

End

Function getPlayerName {

Return this.playerName as String

End

Function getPlayerAge {

Return this.playerAge as Integer

End

Function getPlayerHeight {

Return this.playerHeight.toString as String

End

Function getPlayerHeightInches {

Return this.playerHeight.toInches as Integer

End

Function toString {

Return “Name: “ + this.playerName + “ Age: " + this.playerAge + " Height: " + this.playerHeight.toString as String

End

End

UML Diagram

|  |
| --- |
| **Assignment1** |
| -players: ArrayList |
| -sumOfAges: double |
| -averageAge: double |
| -tallestPlayerInches: int |
| -tallestPlayerIndex: int |
| -count: int |

|  |
| --- |
| **Player** |
| -playerName: String |
| -playerAge: Int |
| -playerHeihgt: Height |
| +Player() |
| +Player(s: String) |
| +getPlayerName(): String |
| +getPlayerAge(): Int |
| +getPlayerHeight(): String |
| +getPlayerHeightInches(): Int |
| +toString(): String |

|  |
| --- |
| **Height** |
| -feet: Int |
| -inch: Int |
| +Height(f: Int, i: Int) |
| +toInches(): Int |
| +toString(): String |

Test Plan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case** | **Input/Output** | **Expected Result** | **Actual Result** | **Outcome (Pass/Fail)** |
| 1a | Prompt user for player Name, Age, and Height [in feet and inches] until input is blank. | User enters a players Name, age, and height (in feet and inches) separated by spaces, enters a blank to end input | While (true) { Output: “Enter the player’s Name, Age, and Height”; String line = inputString.nextLine(); if (line.lenght() == 0) { break; | Pass |
| 2a | Store each player in ArrayList for player class. | Each input is converted to a Player class and added to the ArrayList players | Players.add(new Player(line)); | Pass |
| 3a | Sum the ages of all players and calculate the average age. | Retrieve each player’s age and add together. Divide by number of players | For (Player player : players) { sumOfAges += player.getPlayerAge() }  averageAge = sumOfAges / players.size(); | Pass |
| 4a | Iterate through players and find the age less than the average age. | Retrieve player age and height, compare to averageAge | If (player.getPlayerAge < averageAge) | Pass |
| 5a | Iterate through players and find the tallest height. | Retrieve player height and compare to tallest. Update variables if true | if (player.getPlayerHeight() > tallestPlayerInches) { tallestPlayerIndex = count; tallesPlayerInches = player.getPlayerHeightInches(); } | Pass |
| 6a | Output the Name, Age, and Height of the found player. | Output’s the tallest player whose age is less than the average | Output: “The tallest player whose age is less than the average is “ + players.get(tallestPlayerIndex).toString()); | Pass |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case** | **Input/Output** | **Expected Result** | **Actual Result** | **Outcome (Pass/Fail)** |
| 1 | (1a)  Joe 37 5 10  Stef 40 5 8  Maddie 14 5 6  Carsion 5 3 6 | The tallest player whose age is less than the average is Name: Maddie Age: 14 Height: 5’ 6” | The tallest player whose age is less than the average is Name: Maddie Age: 14 Height: 5’ 6” | Pass |
| 2 | (1a)  Bill 34 6 1  John 42 5 11  Dennis 27 5 10  John 31 5 11 | The tallest player whose age is less than the average is Name: John Age: 31 Height: 5’ 11” | The tallest player whose age is less than the average is Name: John Age: 31 Height: 5’ 11” | Pass |
| 3 | (1a)  Maddie 14 5 6  Miriam 15 5 5  Hayden 15 5 3 | The tallest player whose age is less than the average is Name: Maddie Age: 14 Height: 5’ 6” | The tallest player whose age is less than the average is Name: Maddie Age: 14 Height: 5’ 6” | Pass |
| 4 | (1a)  Carson 5 3 6  Zander 5 3 5  Cory 6 3 9  Zach 6 4 0 | The tallest player whose age is less than the average is Name: Carson Age: 5 Height: 3’ 6” | The tallest player whose age is less than the average is Name: Carson Age: 5 Height: 3’ 6” | Pass |

A computer screen shot of a program

Description automatically generated

A computer screen shot of a program code

Description automatically generated