**Java Journal Template**

**Directions:** Follow the directions for each part of the journal template. Include in your response all the elements listed under the Requirements section. Prompts in the Inspiration section are not required; however, they may help you to fully think through your response.

Remember to review the Touchstone page for entry requirements, examples, and grading specifics.

**Name: Sarah Copeland**

**Date: 07/27/2025**

**Final Replit Program Share Link:** <https://replit.com/@sarahcopeland/OKC-Thunder-Stat-Tracker#src/main/java/Main.java>

## PART 1: Defining Your Problem

|  |
| --- |
| **Task**  State the problem you are planning to solve.  **Requirements**   * Describe the problem you are trying to solve. * Describe any input data you expect to use. * Describe what the program will do to solve the problem. * Describe any outputs or results the program will provide.   **Inspiration**  When writing your entry below, ask yourself the following questions:   * Is your problem clearly defined? * Why do you want to solve this particular problem? * What source(s) of data do you believe you will need? Will the user need to supply that data, or will you get it from an external file or another source? * Will you need to interact with the user throughout the program? Will users continually need to enter data in and see something to continue? * What are your expected results or what will be the end product? What will you need to tell a user of your program when it is complete? |
| For this project, I’m building an Oklahoma City Thunder-themed player statistics tracker that lets a user input and review OKC Thunder player stats from a game. The goal is to make it easy to track individual player performance and get team totals and averages without doing the math manually.  The input will come from the user. First, they’ll say how many players they want to track (between 2 and 5). Then, the program will ask for each player’s name, points, rebounds, and assists.  The program stores this data, does the calculations, and then displays each player’s stats along with the team’s total and average points, rebounds, and assists. |

## PART 2: Working Through Specific Examples

|  |
| --- |
| **Task**  Write down clear and specific steps to solve a simple version of your problem you identified in Part 1.  **Requirements**  Complete the three steps below **for at least two distinct examples/scenarios**.   * State any necessary input data for your simplified problem. * Write clear and specific steps in English (not Java) detailing what the program will do to solve the problem. * Describe the specific result of your example/scenario.   **Inspiration**  When writing your entry below, ask yourself the following questions:   * Are there any steps that you don’t fully understand? These are places to spend more time working out the details. Consider adding additional smaller steps in these spots. * Remember that a computer program is very literal. Are there any steps that are unclear? Try giving the steps of your example/scenario to a friend or family member to read through and ask you questions about parts they don’t understand. Rewrite these parts as clearly as you can. * Are there interesting edge cases for your program? Try to start one of your examples/scenarios with input that matches this edge case. How does it change how your program might work? |
| Scenario 1: Three Players, All With Stats Input: - Number of players: 3 - Player 1: Shai Gilgeous-Alexander – 31 points, 6 rebounds, 5 assists - Player 2: Jalen Williams – 18 points, 4 rebounds, 7 assists - Player 3: Chet Holmgren – 22 points, 10 rebounds, 2 assists  Steps: 1. User enters 3 players 2. Inputs names and stats for each player 3. Program stores all data 4. It sums up team totals:  - Points: 71  - Rebounds: 20  - Assists: 14 5. It calculates averages:  - Avg points: 23.7  - Avg rebounds: 6.7  - Avg assists: 4.7  Scenario 2: Two Players With Zero Stats Input: - Number of players: 2 - Player 1: Player A – 0 points, 0 rebounds, 0 assists - Player 2: Player B – 0 points, 0 rebounds, 0 assists  Steps: 1. User enters 2 players 2. Enters all zeros for each 3. Program stores data 4. Totals and averages are all 0  Output: Confirm that the program handles edge cases correctly. |

## PART 3: Generalizing Into Pseudocode

|  |
| --- |
| **Task**  Write out the general sequence your program will use, including all specific examples/scenarios you provided in Part 2.  **Requirements**   * Write pseudocode for the program in English but refer to Java program elements where they are appropriate. The pseudocode should represent the full functionality of the program, not just a simplified version. Pseudocode is broken down enough that the details of the program are no longer in any paragraph form. One statement per line is ideal.   **Help With Writing Pseudocode**   * Here are a few links that can help you write pseudocode with examples. Remember to check out part 3 of the Example Journal Template Submission if you have not already. Note: everyone will write pseudocode differently. There is no right or wrong way to write it, other than to make sure you write it clearly and in as much detail as you can so that it should be easy to convert to code later.   + <https://www.geeksforgeeks.org/how-to-write-a-pseudo-code/>   + <https://www.wikihow.com/Write-Pseudocode>   **Inspiration**  When writing your entry below, ask yourself the following questions:   * Do you see common program elements and patterns in your specific examples/scenarios in Part 2, like variables, conditionals, functions, loops, and classes? These should be part of your pseudocode for the general sequence as well. * Are there places where the steps for your examples/scenarios in Part 2 diverged? These may be places where errors may occur later in the project. Make note of them. * When you are finished with your pseudocode, does it make sense, even to a person that does not know Java? Aim for the clearest description of the steps, as this will make it easier to convert into program code later. |
| Start Program Display welcome message Prompt user: how many players? (between 2 and 5) Create arrays for player names, points, rebounds, assists For each player:  Ask for name  Ask for points  Ask for rebounds  Ask for assists  Store the values in the arrays Initialize totalPoints, totalRebounds, totalAssists to 0 For each player:  Add player stats to totalPoints, totalRebounds, totalAssists Calculate averagePoints = totalPoints / number of players Calculate averageRebounds = totalRebounds / number of players Calculate averageAssists = totalAssists / number of players Display individual player stats Display total team stats Display average team stats End Program |

## PART 4: Testing Your Program

|  |
| --- |
| **Task**  While writing and testing your program code, describe your tests, record any errors, and state your approach to fixing the errors.  **Requirements**   * For at least one of your test cases, describe how your choices for the test helped you understand whether the program was running correctly or not.   For each error that occurs while writing and testing your code:   * Record the details of the error from Replit. A screenshot or copy-and-paste of the text into the journal entry is acceptable. * Describe what you attempted in order to fix the error. Clearly identify which approach was the one that worked.   **Inspiration**  When writing your entry below, ask yourself the following questions:   * Have you tested edge cases and special cases for the inputs of your program code? Often these unexpected values can cause errors in the operation of your program. * Have you tested opportunities for user error? If a user is asked to provide an input, what happens when they give the wrong type of input, like a letter instead of a number, or vice versa? * Did the outcome look the way you expected? Was it formatted correctly? * Does your output align with the solution to the problem you coded for? |
| Test Case 1: Three Valid Players Input: - Shai: 31 pts, 6 reb, 5 ast - Jalen: 18 pts, 4 reb, 7 ast - Chet: 22 pts, 10 reb, 2 ast  Output: - Team totals: 71 pts, 20 reb, 14 ast - Averages: 23.7 pts, 6.7 reb, 4.7 ast This test confirmed the math and output formatting was correct.  Error 1: InputMismatchException I typed a name where it asked for points by mistake. The program crashed.  Fix: I added input validation using a try/catch block for numeric input. I also included a scanner reset (scanner.nextLine()) after a mismatch to clear the buffer.  After this fix, the program handled bad inputs and showed clean stats as expected. |

## PART 5: Commenting Your Program

|  |
| --- |
| **Task**  Submit your full program code, including thorough comments describing what each portion of the program should do when working correctly.  **Requirements**   * The purpose of the program and each of its parts should be clear to a reader that does not know the Java programming language.   **Inspiration**  When writing your entry, you are encouraged to consider the following:   * Is each section or sub-section of your code commented to describe what the code is doing? * Give your code with comments to a friend or family member to review. Add additional comments to spots that confuse them to make it clearer. |
| import java.util.Scanner; // Import the Scanner class to read user input  // Main class for the Thunder Stats Tracker program  public class Main {  // Main method where the program starts  public static void main(String[] args) {  Scanner scanner = new Scanner(System.in); // Scanner object for reading user input  // Welcome message for the user  System.out.println("Welcome to the OKC Thunder Game Stats Tracker!");  // Prompting user for how many players they want to track  int numPlayers = 0;  while (numPlayers < 2 || numPlayers > 5) {  System.out.print("Enter number of Thunder players to track (2–5): ");  try {  numPlayers = scanner.nextInt(); // Read user input  if (numPlayers < 2 || numPlayers > 5) {  System.out.println("Please enter a number between 2 and 5.");  }  } catch (Exception e) {  System.out.println("Invalid input. Please enter a number.");  scanner.nextLine(); // Clear the invalid input from the scanner buffer  }  }  // Create arrays to hold each player's data  String[] names = new String[numPlayers];  int[] points = new int[numPlayers];  int[] rebounds = new int[numPlayers];  int[] assists = new int[numPlayers];  scanner.nextLine(); // Clear scanner buffer before reading player names  // Loop to collect data for each player  for (int i = 0; i < numPlayers; i++) {  System.out.println("\nPlayer " + (i + 1) + ":");  System.out.print("Name: ");  names[i] = scanner.nextLine(); // Get player's name  // Use helper method to get and validate numeric input  points[i] = getValidIntInput(scanner, "Points: ");  rebounds[i] = getValidIntInput(scanner, "Rebounds: ");  assists[i] = getValidIntInput(scanner, "Assists: ");  }  // Calculate team totals for each stat  int totalPoints = 0, totalRebounds = 0, totalAssists = 0;  for (int i = 0; i < numPlayers; i++) {  totalPoints += points[i];  totalRebounds += rebounds[i];  totalAssists += assists[i];  }  // Calculate averages based on total values and number of players  double avgPoints = (double) totalPoints / numPlayers;  double avgRebounds = (double) totalRebounds / numPlayers;  double avgAssists = (double) totalAssists / numPlayers;  // Display each player's individual stats  System.out.println("\n--- Player Stats ---");  for (int i = 0; i < numPlayers; i++) {  System.out.println(names[i] + ": " + points[i] + " pts, " + rebounds[i] + " reb, " + assists[i] + " ast");  }  // Display the full team summary, including totals and averages  System.out.println("\n--- Team Summary ---");  System.out.println("Total Points: " + totalPoints);  System.out.println("Total Rebounds: " + totalRebounds);  System.out.println("Total Assists: " + totalAssists);  // Print averages formatted to one decimal place  System.out.printf("Avg Points: %.1f\n", avgPoints);  System.out.printf("Avg Rebounds: %.1f\n", avgRebounds);  System.out.printf("Avg Assists: %.1f\n", avgAssists);  // Final message  System.out.println("\nThanks for using the Thunder Stats Tracker!");  // Close the scanner  scanner.close();  }  // This helper method ensures the user enters a valid integer for stats  // It loops until a valid number is entered, helping prevent crashes  public static int getValidIntInput(Scanner scanner, String prompt) {  int value = -1;  boolean valid = false;  while (!valid) {  System.out.print(prompt);  try {  value = scanner.nextInt(); // Try reading the number  valid = true;  } catch (Exception e) {  System.out.println("Invalid input. Please enter a number.");  scanner.nextLine(); // Clear invalid input  }  }  return value;  }  }  In my program, I added comments throughout the code to explain exactly what each part does so that someone without Java experience can still follow the logic. At the very top, I explain that the program is a stats tracker for OKC Thunder players. I also include a comment where the program starts in the main method, so it's clear where the logic begins.  When the user is asked to enter how many players they want to track, I explain that the program uses a while loop and a try/catch block to make sure the input is valid. That helps avoid crashes if the user enters something other than a number. I also comment on the arrays being used to store player names and stats and explain that we use another loop to go through each player and gather their input.  In the part of the code where I calculate totals and averages, I clearly label what’s being added up and how the averages are calculated. I made sure to explain how the results are displayed at the end of the program, both for individual players and the full team summary. Each section is separated with comments that describe the purpose of the block, like calculating totals or printing stats.  I also created a helper method called getValidIntInput, and I added a comment at the top of that method explaining that it’s used to validate user input so the program doesn't crash if someone types letters instead of numbers. I walked through the logic with friends and added extra comments anywhere they seemed unsure about what was happening.  The goal of my comments is to guide someone step-by-step through the entire program even if they’ve never seen Java before. I kept the language simple and made sure to explain each major task the program is doing, such as storing values, handling errors, doing math, and printing results. Everything is labeled in a way that makes the code easy to read and understand. |

## 

## PART 6: Your Completed Program

|  |
| --- |
| **Task**  Provide the Replit link to your full program code.  **Requirements**   * The program must work correctly with all the comments included in the program.   **Inspiration**   * Check before submitting your Touchstone that your final version of the program is running successfully. |
| Here’s the link to the full working version of my OKC Thunder Stats Tracker program:  <https://replit.com/@sarahcopeland/OKC-Thunder-Stat-Tracker#src/main/java/Main.java>  Here is my code again if needed:  public class Main {  public static void main(String[] args) {  Scanner scanner = new Scanner(System.in);  // Welcome message  System.out.println("Welcome to the OKC Thunder Game Stats Tracker!");  // Ask how many players to track  int numPlayers = 0;  while (numPlayers < 2 || numPlayers > 5) {  System.out.print("Enter number of Thunder players to track (2–5): ");  try {  numPlayers = scanner.nextInt();  if (numPlayers < 2 || numPlayers > 5) {  System.out.println("Please enter a number between 2 and 5.");  }  } catch (Exception e) {  System.out.println("Invalid input. Please enter a number.");  scanner.nextLine(); // Clear bad input  }  }  // Create arrays to store data  String[] names = new String[numPlayers];  int[] points = new int[numPlayers];  int[] rebounds = new int[numPlayers];  int[] assists = new int[numPlayers];  scanner.nextLine(); // Clear buffer before reading names  // Gather data for each player  for (int i = 0; i < numPlayers; i++) {  System.out.println("\nPlayer " + (i + 1) + ":");  System.out.print("Name: ");  names[i] = scanner.nextLine();  points[i] = getValidIntInput(scanner, "Points: ");  rebounds[i] = getValidIntInput(scanner, "Rebounds: ");  assists[i] = getValidIntInput(scanner, "Assists: ");  }  // Calculate totals  int totalPoints = 0, totalRebounds = 0, totalAssists = 0;  for (int i = 0; i < numPlayers; i++) {  totalPoints += points[i];  totalRebounds += rebounds[i];  totalAssists += assists[i];  }  // Calculate averages  double avgPoints = (double) totalPoints / numPlayers;  double avgRebounds = (double) totalRebounds / numPlayers;  double avgAssists = (double) totalAssists / numPlayers;  // Display player stats  System.out.println("\n--- Player Stats ---");  for (int i = 0; i < numPlayers; i++) {  System.out.println(names[i] + ": " + points[i] + " pts, " + rebounds[i] + " reb, " + assists[i] + " ast");  }  // Display totals and averages  System.out.println("\n--- Team Summary ---");  System.out.println("Total Points: " + totalPoints);  System.out.println("Total Rebounds: " + totalRebounds);  System.out.println("Total Assists: " + totalAssists);  System.out.printf("Avg Points: %.1f\n", avgPoints);  System.out.printf("Avg Rebounds: %.1f\n", avgRebounds);  System.out.printf("Avg Assists: %.1f\n", avgAssists);  System.out.println("\nThanks for using the Thunder Stats Tracker!");  scanner.close();  }  // Helper method to get valid int input  public static int getValidIntInput(Scanner scanner, String prompt) {  int value = -1;  boolean valid = false;  while (!valid) {  System.out.print(prompt);  try {  value = scanner.nextInt();  valid = true;  } catch (Exception e) {  System.out.println("Invalid input. Please enter a number.");  scanner.nextLine(); // Clear bad input  }  }  return value;  }  } |