Specifications

Problem Description

You are going to write a program that repeatedly obtains the scores of five players in a game until a sentinel value is entered. The program will then compute some statistics on the scores and display a summary.

Input: The program will prompt the user to enter the number of the player or the sentinel value. If the number is a valid player number (between 1 and 5), then the program will prompt the user to enter the number of points the player earned (an integer). Points must be 0 or positive. You may assume the user enters the correct data type at each prompt, but you may not assume that the data falls within a meaningful range. Your program must validate the inputs.

Output: After the user has finished entering data and has entered the sentinel value, if any valid data was entered, then the program will display the total number of points scored by all players (as it is a cooperative game and the players represent a team), the highest scoring player, and a summary table composed of the player number, points earned by the player, and percentage of team points earned by that player (the table will always display all players, regardless of whether they scored points). Percentages will show two decimal places.

Interface: Use the following prompts and output messages (exactly, except for values that should be substituted in, such as player number, points, and percentages, which are underlined):

- `Which player? (1, 2, 3, 4, or 5; -1 to quit):`
- `How many points did player 1 score?`
- `Team Total` 600
- `High Score` 200
- `Star Player` `Player 5`
- `Player` `Points` `% of Total`
  1 100 16.67

Use the following error messages for invalid input or no input:

- `That is not a valid player number.`
- `That is an invalid score.`
- `No scores were entered.`

If the user enters a bad player number, they should see the `Which player?` prompt again. If they enter a bad score, you may either prompt for the score until they give a valid score, or prompt for the player number again. The sample solution is programmed to repeatedly prompt for the player score until a valid score is given.

Once the user has entered the sentinel value, if there is at least one valid non-zero count, the team total, high score, star player, and summary information should be calculated and the summary information should be displayed in a table like the following (substitute actual data and calculated results):

```
Player          Points              % of Total
1               100                 25.00
2               100                 25.00
3               0                   0.00
4               200                 50.00
5               0                   0.00
```

The column width for the for the player column is 15 characters, the column width for the other two columns is 20 characters, and the percentage should be displayed to two decimal places.

If there are no valid non-zero counts, the program should display the following message:

```
No scores were entered.
```

Design specifications:

Use only techniques that are covered in the first three lessons. Use console input and output.

You must write a sentinel-controlled loop to obtain player data, with a sentinel value of -1.

Do not, under any circumstances, use `JOptionPane`, `System.exit` or `return`. Use `break` only in `switch` statements. Do not submit code that includes these constructs (except for `break` in `switch` statements). Submit whatever you can write that does not include these constructs.

Testing

Test Case 1

Purpose
One score entered for each player.

Input
Player – 1, Score – 100
Player – 2, Score – 50
Player – 3, Score – 50
Player – 4, Score – 100
Player – 5, Score – 200
Player – -1

Expected output
Results were calculated mentally.

```
Team Total      500
High Score      200
Star Player     Player 5
Player          Points              % of Total
1               100                 20.00
2               50                  10.00
3               50                  10.00
4               100                 20.00
5               200                 40.00
```

Test Case 2

Purpose

Several scores entered for some players, no scores entered for others. Testing that the program correctly sums values for individual players.

Input

Player – 1, Score – 100
Player – 3, Score – 50
Player – 1, Score – 50
Player – 5, Score – 200
Player – -1

Expected output

Results were calculated mentally.

```
Team Total      400
High Score      200
Star Player     Player 5
Player          Points              % of Total
1               150                 37.50
2               0                   0.00
3               50                  12.50
4               0                   0.00
5               200                 50.00
```

Test Case 3

Purpose

Invalid player values. Invalid count value, mixed with valid player and counts.

Input

Player – 1, Score – 100
Player – 6
Player – 3, Score – 50
Player – 1, Score – -10
Player – 1, Score – -2
Player – 1, Score – 50
Player – 5, Score – 200
Player – -1

Expected output

Results were calculated mentally.

```
Team Total      400
High Score      200
Star Player     Player 5
Player          Points              % of Total
1               150                 37.50
2               0                   0.00
3               50                  12.50
4               0                   0.00
5               200                 50.00
```

Test Case 4

Purpose
Invalid player value. Invalid count value.

Input
Player – 6
Player – 1, Score – -10
Player – 1, Score – 0
Player – -2
Player – -1

Expected output
```
No scores were entered.
```

Test Case 5

Purpose
Sentinel value entered immediately.

Input
Player -1

Expected output
```
No scores were entered.
```

Rubric
An exceptional-quality assignment will meet the following standards:

- Meeting functional and design specifications
  The Java program works and meets all of the specifications, with no additional unspecified functionality. The programmer has used programming techniques from the first through third lessons only. A sentinel loop is used to obtain data. If the program misses specifications or does not function correctly, errors are acknowledged with a thorough and reflective analysis in the testing section (points will be removed for missed specifications).
- Communicating with identifiers and white space
  The program makes appropriate use of variables. Variables and constants are named according to convention and are named for understandability and purpose. White space, both vertical and horizontal, is correctly used for readability and meets programming conventions.
- Communicating through documentation
  The Java program contains comments including the programmer's name and date. There are block comments (as many as necessary) for each distinct block of code which accurately describe what the block is accomplishing by relating the code to the problem being solved. Javadoc is included and meets the javadoc standards.
- Assumptions and Testing
  Testing is thorough. If there are errors, they are described in the testing section. If there are

questions, they are answered thoughtfully in the testing section. All assumptions are made explicit.