|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **LEARNING PROFILE FOR ASSIGNMENT#\_1\_\_\_\_\_ AND QUESTION#\_1\_\_\_\_\_\_\_** | | | | | |
| *Name* | *:* | *Steven Morrissey* | *Due Date* | *:* |  |
| *Student ID* | *:* | *3300222* | *Submission Date* | *:* |  |

**1. Problem Statement:**

Read three sentences from the console application. Each sentence should not exceed 80 characters. Then, copy each character in each input sentence in a [3 x 80] character array. The first sentence should be loaded into the first row in the reverse order of characters – for example, “mary had a little lamb” should be loaded into the array as “bmal elttil a dah yram”. The second sentence should be loaded into the second row in the reverse order of words – for example, “mary had a little lamb” should be loaded into the array as “lamb little a had mary”. The third sentence should be loaded into the third row where if the index of the array is divisible by 5, then the corresponding character is replaced by the letter ‘z’ – for example, “mary had a little lamb” should be loaded into the array as “mary zad azlittze lazb” – that is, characters in index positions 5, 10, 15, and 20 were replaced by ‘z’. Note that an empty space is also a character, and that the index starts from position 0. Now print the contents of the character array on the console.

**2. Description of the Code:**

**3. Errors and Warnings:**

Table 1: List of Errors and Warnings Encountered in the Program

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Errors / Warnings** | **Details** | **How I solved them** |
| 1 | none |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  |  |
| 5 |  |  |  |

**4. Sample Input and Output:**

**5. Discussion:**

The question did not make sense to me in the beginning, as it’s not how I would structure an “Address Book” class myself (returning String for compare names, having no data structure fields to hold data sets instead of single values, etc…). I searched the forum and found other people with similar confusion and gained some understanding from their responses. I also found discrepancies between the API and the question, as it looks like they are passing in a name for getMiddleName which isn’t specified in the API.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **LEARNING PROFILE FOR ASSIGNMENT#\_1\_\_\_\_\_ AND QUESTION#\_2\_\_\_\_\_\_\_** | | | | | |
| *Name* | *:* | *Steven Morrissey* | *Due Date* | *:* |  |
| *Student ID* | *:* | *3300222* | *Submission Date* | *:* |  |

**1. Problem Statement:**

Write a program that plays the Rock-Paper-Scissors-Lizard-Spock game. Refer to http://en.wikipedia.org/wiki/Rock-paper-scissors-lizard-Spock for more information. Normally, one player is a human and the other is the computer program. However, in this exercise, the program will generate two players who play against each other. The play continues until either of the computer-generated players wins four consecutive times. In this game, two random integers are generated in the range of [1 to 5], one per player. 1 refers to Rock, 2 refers to Paper, 3 refers to Scissors, 4 refers to Lizard, and 5 refers to Spock. For example, if the computer randomly generates integers 2 and 5 in the first iteration, 2 is for the first player and 5 is for the second player. Based on Rule 8 in the following 10 rules, Paper (2) disproves Spock (5), so Player 1 wins. Repeat it to generate one more pair and determine who wins that iteration. Continue the iterations until one player wins four consecutive times.

**2. Description of the Code:**

Strategy was to create a list of all quarterlies, then iterate through to have less repeating code. Finally calculate annual bonus and add it to the running total to be returned at the end of the method.

**3. Errors and Warnings:**

Table 1: List of Errors and Warnings Encountered in the Program

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Errors / Warnings** | **Details** | **How I solved them** |
| 1 | E1 | Forgot the ‘>’ before the ‘=’ in one of the if statements | Found at compile time and place the ‘>’ in the logic |
| 2 | E2 | Used number formatter in the return statement of the method – NumberFormatExcetion. | Moved the formatter to the main method where it can return a String and I don’t have to parse a double |

**4. Sample Input and Output:**

**Case 1: create new object and call computeBonus(2000, 5000, 7000, 4000, 8000), expecting 1,650 as the printed out value of total bonus**

**Case 2: create new object and call computeBonus(2000, 6000, 9000, 10000, 17000), expecting 3,180 as the printed out value of total bonus**

**5. Discussion:**

None, it was straight forward.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **LEARNING PROFILE FOR ASSIGNMENT#\_1\_\_\_\_\_ AND QUESTION#\_3\_\_\_\_\_\_\_** | | | | | |
| *Name* | *:* | *Steven Morrissey* | *Due Date* | *:* |  |
| *Student ID* | *:* | *3300222* | *Submission Date* | *:* |  |

**1. Problem Statement:**

**Credit card numbers follow certain patterns. A credit card number must have between 13 and 16 digits. It must start with 4 for Visa cards, 5 for Master cards, 37 for American Express cards, and 6 for Discover cards. In 1954, Hans Luhn of IBM proposed the following algorithm for validating credit card numbers: a. Double every second digit from right to left (e.g., if number is 3 => 3 \* 2 => 6) and add them together. b. If this doubling results in a two-digit number, then add the two digits to get a single-digit number (e.g., if number is 5 => 5 \* 2 => 10 => 1+0 => 1). So, for the credit card number 4388576018402626, doubling all second digits from the right results in (2 \* 2 = 4) + (2 \* 2 = 4) + (4 \* 2 = 8) + (1 \* 2 = 2) + (6 \* 2 = 12 = 1 + 2 = 3) + (5 \* 2 = 10 = 1 + 0 = 1) + (8 \* 2 = 16 = 1 + 6 = 7) + (4 \* 2 = 8). This totals to 4 + 4 + 8 + 2 + 3 + 1 + 7 + 8 = 37. Add all digits in the odd places from right to left. The leftmost digit of the credit card number is at index 0; 6 + 6 + 0 + 8 + 0 + 7 + 8 + 3 = 38. Add results from steps (a) and (b) and see if divisible by 10. If it is, then the card number is valid; otherwise invalid. 37 + 38 = 75 is not divisible by 10, so it is an invalid credit card number. Implement Luhn’s algorithm in a program to determine whether a given credit card number is valid or not. You must test if the number of digits in the input is in the valid range (13 to 16), run Luhn’s algorithm to test its validity, and if it is valid, print the name of the company that offers that credit card number.**

**2. Description of the Code:**

**3. Errors and Warnings:**

Table 1: List of Errors and Warnings Encountered in the Program

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Errors / Warnings** | **Details** | **How I solved them** |
| 1 | none |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  |  |
| 5 |  |  |  |

**4. Sample Input and Output:**

**5. Discussion:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **LEARNING PROFILE FOR ASSIGNMENT#\_1\_\_\_\_\_ AND QUESTION#\_4\_\_\_\_\_\_\_** | | | | | |
| *Name* | *:* | *Steven Morrissey* | *Due Date* | *:* |  |
| *Student ID* | *:* | *3300222* | *Submission Date* | *:* |  |

**1. Problem Statement:**

Create a 10x10 matrix as a 2D array. Assume that a robot is placed in position [0, 0]. Now randomly generate a move. The move could take the robot to one of the eight possible adjacent slots – {up, down, left, right, left-upcorner, left-down-corner, right-up-corner, and right-down-corner} – these slots are represented by {1, 2, 3, 4, 5, 6, 7, 8}. However, at [0, 0], the robot only has three possible slots to move to – right, down, right-down-corner. Create another robot called R2 and place it on [9, 9]. Now randomly generate an integer in the range of [1 to 8]. This first random integer corresponds to a possible move for Robot R1. If the move is valid, then move R1 to its new slot. A move is invalid if it takes the robot out of bounds of the [10x10] matrix. If the move is invalid, then keep generating random integers until a valid move is found. Repeat this procedure for the second Robot R2. If both R1 and R2 are in the same slot, then stop, print the final slot, print the sequence of random numbers that led R1 to this slot, and the print the sequence of random numbers that led R2 to the same slot. Implement this program with a Robot class and a MovingRobot subclass.

**2. Description of the Code:**

**3. Errors and Warnings:**

Table 1: List of Errors and Warnings Encountered in the Program

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Errors / Warnings** | **Details** | **How I solved them** |
| 1 | none |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  |  |
| 5 |  |  |  |

**4. Sample Input and Output:**

**5. Discussion:**