Seneca College

Jan 16, 2019

Applied Arts & Technology

SCHOOL OF COMPUTER STUDIES

JAC444 **Demo Due date : Jan 25, 2019**

Final Code Submission Date: Jan 25, 2019

Workshop 2

Notes:

- **i.** Each task should be presented during the lab, demo worth 70% of the workshop marks and code uploading worth the other 30%.
- **ii.** All the tasks should be demoed in Jan 23rd lab.
- **iii.** Make sure you have all security and check measures in place, like wrong data types etc., no need to implement Exception as we haven't covered yet (If you know it you are allowed to use). There are other ways to handle bad input data.
- **iv.** Given output structure is just for student to have a glimpse what the output can look, student are free to make the output better in any way.

Other inputs can be given during demo, so make sure you test your program properly.

Task 1: Write a hangman game that randomly generates a word and prompts the user to guess one letter at a time, as shown in the sample run. Each letter in the word is displayed as an asterisk. When the user makes a correct guess, the actual letter is then displayed. When the user finishes a word, display the number of misses and ask the user whether to continue to play with another word. Declare an array to store words, as follows:

```
// Add any words you wish in this array
String[] words = {"write", "that", ...};
```

```
(Guess) Enter a letter in word ******* > p Enter
(Guess) Enter a letter in word p****** > r Enter
(Guess) Enter a letter in word pr**r** > p Enter
     p is already in the word
(Guess) Enter a letter in word pr**r** > o Enter
(Guess) Enter a letter in word pro*r** > g Enter
(Guess) Enter a letter in word progr** > n Enter
     n is not in the word
(Guess) Enter a letter in word progr** > m Enter
(Guess) Enter a letter in word progr*m > a Enter
The word is program. You missed 1 time
Do you want to guess another word? Enter y or n>
```

Task 2: Design a class named **Location** for locating a maximal value and its location in a two-dimensional array. The class contains data field's **row**, **column**, and **maxValue** that store the maximal value and its indices in a two-dimensional array with **row** and **column** and **maxValue** as a **double** type.
Write the following method that returns the location of the largest element in a two-dimensional array:

```
public static Location locateLargest(double[][] a)
```

The return value is an instance of **Location**. Write a test program that prompts the user to enter a two-dimensional array and displays the location of the largest element in the array. Here is a sample run:

```
Enter the number of rows and columns in the array:  3 4 Enter
Enter the array:
23.5 35 2 10 Enter
4.5 3 45 3.5 Enter
35 44 5.5 9.6 Enter
The location of the largest element is 45 at (1, 2)
```

Task 3: 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
- 6 for Discover cards

In 1954, Hans Luhn of IBM proposed an algorithm for validating credit card numbers. The algorithm is useful to determine whether a card number is entered correctly or whether a credit card is scanned correctly by a scanner. Credit card numbers are generated following this validity check, commonly known as the *Luhn check* or the *Mod 10 check,* which can be described as follows (for illustration, consider the card number 4388576018402626):

1. Double every second digit from right to left. If doubling of a digit results in a two-digit number, add up the two digits to get a single-digit number.
2. Now add all single-digit numbers from Step 1.
   4 + 4 + 8 + 2 + 3 + 1 + 7 + 8 = 37
3. Add all digits in the odd places from right to left in the card number.
   6 + 6 + 0 + 8 + 0 + 7 + 8 + 3 = 38
4. Sum the results from Step 2 and Step 3.
   37 + 38 = 75
5. If the result from Step 4 is divisible by 10, the card number is valid; otherwise, it is invalid. For example, the number 4388576018402626 is invalid, but the number 4388576018410707 is valid.

Write a program that prompts the user to enter a credit card number as a **long** integer (can also use string to solve this). Display whether the number is valid or invalid. Design your program to create and use the following methods:

1. A method which should: Return true if the card number is valid
2. A method which should: Get the result from Step 2
3. A method which should: Return this number if it is a single digit, otherwise, return the sum of the two digits
4. A method which should: Return sum of odd-place digits in number
5. A method which should: Return true if the digit is a prefix for number
6. A method which should: Return the number of digits
7. A method which should: Return the first k number of digits from number. If the number of digits in number is less than k, return number.

Here are sample runs of the program:

```
Enter a credit card number as a long integer:
  4388576018410707 Enter
4388576018410707 is valid
```

```
Enter a credit card number as a long integer:
  4388576018402626 Enter
4388576018402626 is invalid
```