Major steps for solving the problem:

This project was somewhat tricky. You needed to read numbers from right to left, which is not usually easy programmatically. Here are the major steps:

1. Check if the prefix (or first number) matches preset prefixes for the major credit card companies. Visa is 4, MasterCard is 5, Amex is 37, and Discover is 6. If the prefix matched, go to the next step, and if it doesn’t, then it’s not valid.
2. Then you’d test if the card number was not too short or too long. Credit card numbers are as short as 13 numbers and as long as 16 numbers. Anything less than 13 or more than 16, it’s not valid.
3. Then you need to do add up every other number in the evens place from right to left after multiplying it by 2. If the answer is a 2 digit number, add the numbers together to get a 1 digit number.
4. Now do the same with the odds place, but do not multiply the number by 2, just add up all the numbers in the odds place.
5. Add the two numbers you have.
6. If the number is divisible by 10, then the card is valid. If the number is not divisible by 10, then it’s not valid.

Pseudo Code for CreditCard.java

1. Main method:
   1. Prompt the user to enter a credit card number
   2. Get data from calling inputData() and show the result by passing the result from the previous method to displayResult()
   3. Print the programmer’s name after the result is displayed
2. inputData method:
   1. Create new Scanner object for input
   2. Get input from user and store it in a variable of type long.
   3. Close the input to not leak memory
   4. Return the long that you got from the user.
3. displayResult method:
   1. Output if the credit card if valid or not by calling isValid()
4. isValid method:
   1. Check if the size of the number is within 13 and 16 inclusive, calling getSize()
      1. If true, create an array that holds the correct prefixes for all the credit cards (4, 5, 37, 6)
      2. Loop through the array of prefixes
         1. For every item, check if the prefix matches, calling prefixMatched()
            1. If it matches, get the sums of the numbers from evens place and odds place, then add it together by calling sumOfDoubleEvenPlace() and sumOfOddPlace().
            2. Check if the number is divisible by 10

If true, return true

* + 1. Return false if everything is looped through and a match is not found
  1. Return false because the size didn’t match.

1. getSize method:
   1. Return the length of the d
2. prefixMatched method:
   1. Get the size of d, calling getSize()
   2. Convert d into a String to take advantage of the String.equals() method.
   3. Check if the length of d is 2
      1. If it is two:
         1. Get the prefix by calling getPrefix()
         2. Return true if the prefix matches the number that the isValid method provided
         3. Return false if the prefix does not match the number that the isValid method provided.
      2. If it’s 1:
         1. Get the prefix by caling getPrefix().
         2. Return true if the prefix matches the number that the isValid method provided
         3. Return false of the prefix does not match the number that the isValid method provided.
   4. Return false if the number is neither 1 or 2, but that should never happen.
3. sumOfDoubleEvenPlace method:
   1. Reverse the number so you can do stuff with it from right to left.
   2. Create an array that will hold the numbers in the evens place.
   3. Create a int sum that will hold the sum of the evens place numbers
   4. Do a loop with a counter k that goes up by 1 and another counter i by 2 every iteration. Loop until i is less than the length of the reversed number where i is 1 and k is 0
      1. Set the index k of the doubles array to the reversed number array at index i.
   5. Loop through every number in the doubles array
      1. Add result of getDigit() to sum, with the number multiplied by 2 passed as a parameter If the number is not null.
   6. Return the sum
4. sumOfOddPlace method:
   1. Reverse the number so you can do stuff with it from right to left.
   2. Create an array that will hold the numbers in the evens place.
   3. Create a int sum that will hold the sum of the evens place numbers
   4. Do a loop with a counter k that goes up by 1 and another counter i by 2 every iteration. Loop until i is less than the length of the reversed number where i is 0 and k is 0
      1. Set the index k of the doubles array to the reversed number array at index i.
   5. Loop through every number in the doubles array
      1. Add the number to sum if the number is not null.
   6. Return the sum
5. getPrefix method:
   1. Convert the number into a string to take advantage of charAt()
   2. Check if k is 2 (where k is the number I put if the prefix has 2 numbers in it):
      1. If it does, return the concatenation of the strings of charAt(0) and charAt(1). So if the number is 51xxxxx…., then it will do “5” + “1” which is 51.
   3. If it’s 1, then return whatever is at charAt(0)
   4. It should never be something other than 1 or 2 other than programmer error, but just in case, it returns -1 (which is a prefix that is not possible!)
6. getDigit method:
   1. Convert the number into a string to take advantage of String.length()
   2. Check if the length is 2
      1. If it is:
         1. Create sum of type int to hold the sum of the 2 numbers
         2. Split the number by “” delimiter and put into an array
         3. Loop through the array for every element
            1. Add the number into sum
         4. Return the sum.
      2. If not:
         1. Return the number since you don’t need to add 2 numbers and it’s not possible to get a number with more than 2 digits when the number is 1-9 and multiplied by 2 (9x2 = 18).

A picture containing text, map

Description automatically generatedFlow chart:

A close up of a map

Description automatically generated

A close up of a map

Description automatically generated

Test plan:

I found a Lunh’s calculator online to get good numbers that I can pass through my program. It gives me valid numbers, and I can check if my program comes out as valid. I then think of a bunch of random numbers and make sure they are not valid online. Then I pass it through my program. Here are the tests I did:

Test cases:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Case # | Input | Expected Output | Actual Output | Did it pass? |
| Case 1 | Enter a credit card number as a long integer: 4909573084785 | 4909573084785 is valid. | 4909573084785 is valid. | Y |
| Case 2 | Enter a credit card number as a long integer: 4532357865432313 | 4532357865432313 is invalid. | 4532357865432313 is invalid | Y |
| Case 3 | Enter a credit card number as a long integer: 5651655952929471 | 5651655952929471 is valid. | 5651655952929471 is valid. | Y |
| Case 4 | Enter a credit card number as a long integer: 62812692532297 | 62812692532297 is valid. | 62812692532297 is valid. | Y |

Screenshot 1:

A screenshot of a cell phone

Description automatically generated

Screenshot 2:

A screenshot of a cell phone

Description automatically generated

Screenshot 3:

A screenshot of a cell phone

Description automatically generated

Screenshot 4:

A screenshot of a social media post

Description automatically generated

UML Diagram:

A screenshot of a cell phone

Description automatically generated

Lessons Learned:

I learned the importance about using methods and why you need to use so many. If all this was done in the main code, it would have been very hard to see what was going on as the program itself is over 200 lines of code. It makes it very easy to see what is happening at what time and how things are working. I also learned that certain data types have properties that you may need. Something I used a lot was String.length() and String.split(“”) after converting the number from an integer to a string, by using String.valueOf(number).

**Check List**

|  |  |  |  |
| --- | --- | --- | --- |
| **#** |  | **Y/N** | **Comments** |
|  | **Source java files** | **Y** |  |
|  | **Compressed files:** | **Y** |  |
|  | FirstInitialLastName\_Project4\_Moss.zip | **Y** |  |
|  | FirstInitialLastName\_Project4\_doc.zip | **Y** |  |
|  | **Program compiles** | **Y** |  |
|  | **Program runs** | **Y** |  |
|  | **Checklist is completed and included in the Documentation** | **Y** |  |
|  | **Documentation file:** | **Y** |  |
|  | **Comprehensive Test Plan** | **Y** |  |
|  | **Screenshots based on Test Plan** | **Y** |  |
|  | **UML Diagram** | **Y** |  |
|  | **Algorithms/Pseudocode** | **Y** |  |
|  | **Flowchart** | **Y** |  |
|  | **Lessons Learned** | **Y** |  |