1 Problem Specification:

A common requirement for software applications supporting inventory management is to validate the items being manipulated. For example, suppose a store manager wants to add a new item to the store’s inventory database. To do so, the manager would need to insert a new entry into the database that gives the name and price of the item along with its code number. Correctly entering all three pieces of data is important, since an improperly entered code would make it impossible for price scanners to recognize the item.

One way to increase the likelihood of correct data entry is to use checksum software to test whether the digits of a potential code number have some particular property.

For example, there is a verification scheme for the Universal Product Code (UPC) numbers. The familiar UPC in bar code form is shown below.

![UPC bar code](image)

A valid UPC number is a 12-digit value that meets the following criteria:

1. Let \( m \) be the sum of the 2nd, 4th, 6th, 8th, and 10th digits (where the numbering of digits is from left to right.)
2. Let \( n \) be the sum of the 1st, 3rd, 5th, 7th, 9th, and 11th digits.
3. Let \( r = 10 - ((m + 3 \times n) \mod 10) \)
4. A number meets the UPC encoding criteria if its 12th (last) digit equals \( r \).

Consider the number 780070121354.

For this number,
\[
\begin{align*}
    m &= 8 + 0 + 0 + 2 + 3 = 13 \\
    n &= 7 + 0 + 7 + 1 + 1 + 5 = 21 \\
    r &= 10 - ((13 + 3 \times 21) \mod 10) \\
       &= 10 - (76 \mod 10) \\
       &= 10 - 6 \\
       &= 4
\end{align*}
\]

Because \( r = 4 \), the number corresponds to a possible UPC code.

Now consider the number 123456789011.

For this number,
\[
\begin{align*}
    m &= 2 + 4 + 6 + 8 + 0 = 20 \\
    n &= 1 + 3 + 5 + 7 + 9 + 1 = 26 \\
    r &= 10 - ((20 + 3 \times 26) \mod 10) \\
       &= 10 - (98 \mod 10) \\
       &= 10 - 8 \\
       &= 2
\end{align*}
\]

Because \( r \) is not equal to 1, the 12th digit, the number does not correspond to a possible UPC code.
2 Programming Assignment:

Write a program that does the following:

1. Display the function of the program to the user, i.e., let them know what they should enter and what to expect for results (see example runs in Section 3).

2. Prompt the user to enter a 12-digit number. This is easiest to read as a String.

3. While the number has the wrong number of digits:
   
   Report an invalid entry and ask the user to re-enter the number. This while should continue until a String of numbers of length 12 has been entered.

4. Create a method called getNumberArray, called by the main method, that takes the String as input and extracts the digits into an array of single digits using the charAt method of the String class and the getNumericValue method of the Character class. This method should return an array of ints to the main method.

5. Create a method called computeSum, called by the main method, that consumes the array of ints returned by the getNumberArray method as input and calculates m, n, and r as described above. This method should return a boolean to the main method that is true if r equals the 12th digit and false if r is not equal to the 12th digit.

6. If r equals the 12th digit:
   
   Tell the user that the number is a valid UPC code

7. Else:

   Tell the user that the number is not a valid UPC code

You should make your program as modular as possible and not write all the code in the main method. One way to structure the program is given in the algorithm above. You can write the program in many different ways.

To make steps 1-3 less complex, assume the user will enter only digits 0 through 9 and save the numbers in a String. We have covered a String method that returns the number of characters in a String and one that returns each separate character. In the Character class, there is a method getNumericValue that will convert a char to the equivalent whole number.

You can read the input from the keyboard and display the output using either the classes TextIO, the java.util.Scanner class, or by using methods of the javax.swing.JOptionPane class. Stick to one style of input and output throughout the program.

3 Sample outputs:

Run #1: Valid UPC code entered.

The purpose of this program is to validate UPC codes using a checksum scheme. You will be asked to enter a 12-digit number found on the product bar code. This program will perform the checksum on the digits of the number entered, report whether the number is a valid UPC code.

Please enter a 12-digit UPC code: [780070121354]

780070121354 is a valid UPC code.
The purpose of this program is to validate UPC codes using a checksum scheme. You will be asked to enter a 12-digit number found on the product bar code. This program will perform the checksum on the digits of the number entered, report whether the number is a valid UPC code.

Please enter a 12-digit UPC code: [123123123123]

123123123123 is an invalid UPC code.

Please enter a 12-digit UPC code: [7888779]

Sorry, 7888779 is not a valid code.

Please enter a 12-digit UPC code: [yrwkd]

Sorry, yrwkd is not a valid code.

Please enter a 12-digit UPC code: [128016691675]

128016691675 is a valid UPC code.

4 What you should hand in:

When your program compiles and runs correctly, submit all the files on Moodle.

Your program should include a descriptive header comment and should have either intuitively named variables or should contain comments explaining obscure parts of the code.

Be sure to document your program as shown in class and lecture and provide some example test cases in the program header for full credit.