INSTRUCTION:

Submit your solution for this lab to Spectrum latest by next week's lab. Your submission should be in the form of a .zip file (named "Lab8-yourName-yourMatricNum.zip") comprising a .docx file with your answer for Question 1, and a Netbeans project folder containing your source code. State your name, matric number (for e.g. 12345678/9) and tutorial group in your file(s).

Design and implement a Java application to check the validity of a given credit card. You must apply ***Template Method*** Design Pattern.

For simplicity, you only need to consider three types of credit cards — Visa, MasterCard and Diners Club. The application carries out a series of validations on the credit card information. The table below lists different steps in the process of validating different credit cards. As can be seen from the table, some steps of the validation algorithm are the same across all three of the credit cards while some are different. To make sure that the account is in good standing, a check with the credit card company (Visa, MasterCard, Diners Club) is required. This step requires custom programming to interface with the credit card company database and is considered to be different for different credit card types.

Step	*Check*	*Visa*	*MasterCard*	*Diners Club*
1	Expiration date	>Today	>Today	>Today
2	Length	13, 16	16	14
3	Prefix	4	51 through 55	30, 36, 38
4	Valid characters	0 through 9	0 through 9	0 through 9
5	Check digit algorithm	Mod 10	Mod 10	Mod 10
6	Account in good standing	Use custom Visa API	Use custom MasterCard API	Use custom Diners Club API

1. Apply ***Template Method*** Design Pattern to design your system. Draw a UML class diagram to show your design for the system.

   Also, state the following:
   a) Which method is the template method that outlines the validation algorithm?
   b) Which of the steps in the table have the same implementation regardless of the types of credit card and should be implemented as concrete methods in the class containing the template method?
   c) Which of the steps in the table have different implementations for different types of credit card and should be implemented as abstract methods in the class containing the template method?

2. Implement your design in Java. Test your implementation and produce the output below.

```
Card type = VisaCard, Card number = 1234123412341234, Expiry
month = 11, Expiry year = 2004
 Invalid Expiry Date.
```

```
 Invalid Prefix.
 Invalid Check Sum.
This credit card is invalid.

Card type = VisaCard, Card number = 1234567890123456, Expiry
month = 11, Expiry year = 2020
 Invalid Prefix.
 Invalid Check Sum.
This credit card is invalid.

Card type = VisaCard, Card number = 4234567890123456, Expiry
month = 11, Expiry year = 2020
This credit card is valid.

Card type = MasterCard, Card number = 5448755330349315, Expiry
month = 4, Expiry year = 2021
This credit card is valid.

Card type = DinersCard, Card number = 30263720264678, Expiry
month = 5, Expiry year = 2025
This credit card is valid.
```

Hints:
For simplicity, just return **true** in the method that checks whether an account is in good standing as below:

```
public boolean isAccountInGoodStand() {
      /*
      Make necessary API calls to
      perform other checks.
      */
      return true;
}
```

Checksum Algorithm: Mod 10 Check Digit Algorithm
In general, a check digit is a digit added to a number that helps in checking the authenticity of the number. The Mod 10 check digit algorithm can be used to validate such a number associated with a check digit.

The following steps describe the validation process:
1. Use 194774915 (check digit 5 included) as an example.
1 9 4 7 7 4 9 1 5
2. Starting from the second digit from right, multiply every alternate digit by 2.
1 9x2 4 7x2 7 4x2 9 1x2 5
Result:
1 18 4 14 7 8 9 2 5
3. Add individual digits in the newly formed products.
1 1+8 4 1+4 7 8 9 2 5
Result:
1 9 4 5 7 8 9 2 5
4. Now sum up all digits in the resultant number from the above step.
1 +9 +4 +5 +7 +8 +9 +2 +5 = 50
5. Now divide the sum by 10.

Result:
50/10 leaves no remainder and a zero remainder proves that the number is valid.

```
private boolean isValidCheckSum() {
    boolean result = true;

    int sum = 0;
    int multiplier = 1;
    int strLen = cardNum.length();

    for (int i = 0; i < strLen; i++) {
      String digit = cardNum.substring(strLen - i - 1, strLen - i);
      int currProduct = new Integer(digit).intValue() * multiplier;
      if (currProduct >= 10)
        sum += (currProduct % 10) + 1;
      else
        sum += currProduct;
      if (multiplier == 1)
        multiplier++;
      else
        multiplier--;
    }
    if ((sum % 10) != 0)
      result = false;

    return result;
  }
```