SEW I - Übung: Kreditkartennummer-Prüfung

Ziel

Implementierung eines simplen Programms zur Überprüfung von Kreditkartennummern.

Lernziele

- Arbeiten mit Methoden.
- Arbeiten mit *Strings*, *chars* und der *ASCII*-Tabelle.

Abgaberichtlinien

- Ihre implementierte Lösung als `.java`-Datei. **Vergessen Sie dabei nicht auf Kommentare und Kommentarkopf!**
- Achten Sie auf **saubere Variablenbenennung** und **Nutzung von Konstanten** wenn sinnvoll!

Aufgabe

Hintergrund

Eine Kreditkartennummer besteht aus 16 Stellen, wobei die letzte Stelle eine *Prüfziffer* darstellt. Diese soll gegen Fehleingaben schützen und wird folgendermaßen aus den anderen Stellen berechnet:

- Die Ziffern an den Stellen mit gerader Nummer (**beginnend bei 0**) werden verdoppelt und **deren Ziffernsummen** aufsummiert.
- Die Ziffern an den Stellen mit ungerader Nummer werden aufsummiert. Hier muss natürlich keine Ziffernsumme berechnet werden.
- Die auf diese Weise erhaltenen Zahlen werden aufsummiert.
- Die Differenz dieser Summe zur nächsten Zehnerzahl ergibt die Prüfziffer.

Folgendes Beispiel zeigt, wie die Prüfziffer für die Kreditkarten-Nummer 4683 4578 2937 6522 berechnet wird:

Nummer	4	6	8	3	4	5	7	8	2	9	3	7	6	5	2	
Faktor	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	
Produkte	8	6	16	3	8	5	14	8	4	9	6	7	12	5	4	
Ziffernsumme	8	6	7	3	8	5	5	8	4	9	6	7	3	5	4	
Summe																88
Summe *mod* 10																8
Differenz zu 10																**2**

Implementierung

Zu erstellen ist ein Programm, das in der Lage ist Kreditkartennummern auf ihre Gültigkeit zu überprüfen. **Implementieren Sie dabei zuerst die folgenden Methoden und testen Sie diese mit Testdaten durch Aufrufe aus Ihrer `main`-Methode.** Sobald Sie die Richtigkeit der einzelnen Methoden sichergestellt haben, können Sie diese verwenden um das finale Programm (siehe Beispiel unten) inklusive Benutzereingaben zu realisieren.

- `containsOnlyDigits` akzeptiert eine Zeichenkette als Parameter und gibt zurück (`boolean`), ob diese ausschließlich aus Ziffern besteht.
- `calculateDifferenceToNextTen` akzeptiert eine Zahl als Parameter, berechnet die Differenz zur nächsten Zehnerzahl und gibt diese zurück.
- `calculateCheckDigit` akzeptiert eine Zeichenkette und berechnet auf Basis der oben beschriebenen Vorgangsweise die Prüfziffer, die anschließend zurückgegeben wird. Achten Sie auf Code-Verdopplung, verwenden Sie die bisher implementierten Methoden!
- `isCreditCardNumberValid` akzeptiert eine Kreditkartennummer als Zeichenkette und überprüft, ob diese gültig ist. Verwenden Sie dafür Ihre Methode `calculateCheckDigit`, an die Sie die ersten 15 Stellen der Kreditkartennummer übergeben. Sie können die `String`-Methode `substring` verwenden um die Zeichenkette aufzuteilen (siehe unten).
  - Vergessen Sie natürlich nicht zu überprüfen, ob die Länge der Zeichenkette korrekt ist und diese ausschließlich aus Ziffern besteht.

Beispiel

```
==============================
= Credit Card Number Checker =
==============================
Please enter your 16-digit credit card number (or nothing to quit): 1234
Your credit card number is invalid.
Please enter your 16-digit credit card number (or nothing to quit): 123a456b789c123d
Your credit card number is invalid.
Please enter your 16-digit credit card number (or nothing to quit): 4683457829376522
Your credit card number is valid.
Please enter your 16-digit credit card number (or nothing to quit): 0000197419741974
Your credit card number is valid.
Please enter your 16-digit credit card number (or nothing to quit): 4977949494949497
Your credit card number is valid.
Please enter your 16-digit credit card number (or nothing to quit): 4977949494949492
Your credit card number is invalid.
Please enter your 16-digit credit card number (or nothing to quit):
Thank you for using this service!
```

Hinweise

`substring`

Die `substring`-Methode eines `String` akzeptiert einen Start-Index (*inklusiv*) und einen End-Index (*exklusiv*) und gibt die dazwischen liegende Zeichenkette zurück. Beispiel:

```
String demo = "hallowelt";
System.out.println(demo.substring(3,7)); //lowe
```