2. Praktikum: Zahlenpalindrome

Bearbeitungszeitraum:	14.11.22 - 28.11.22, 9.00 Uhr
Gesamtpunktzahl:	5 Abgabepunkte + 10 Leistungspunkte (+5 Bonuspunkte)
Praktikumsziel:	Sie können Ihre Kenntnisse zu Anweisungen und Datentypen zur Umsetzung einer komplexen Aufgabenstellung eigenständig anwenden.
Voraussetzungen:	einfache *Java*-Anweisungen, Datentyp `boolean`, bedingte und zyklische Anweisungen, Umgang mit Java-API-Dokumentation, ggf. Felder in Java
Aufgabe:	Implementieren Sie eine Klasse **Nolindrom**, die alle Zahlen bis zu einer vorgegebenen Größe ermittelt, für die der u. a. Algorithmus (vermutlich) nicht terminiert.
Abgabe:	Die vollständig implementierte Klasse **`Nolindrom.java`** senden Sie wie üblich bei **Moodle** als Lösung ein. *Bitte verwenden Sie genau diesen Programmnamen und achten Sie auf Groß- und Kleinschreibung.* ***ACHTUNG: Ab dieser Aufgabe gibt es auch im Praktikum nur noch Gruppenabgaben.***

Bilden Sie den folgenden Algorithmus in Java nach:

Für eine beliebige natürliche Zahl **N** (in Dezimalschreibweise) sei **R** die Zahl, die sich durch 'Rückwärtslesen' der Ziffernfolge von N ergibt. **N** und **R** werden nun addiert. Ist die Summe eine Palindromzahl (eine Zahl, die vorwärts und rückwärts gelesen den gleichen Wert ergibt), ist der Algorithmus beendet. Falls die Summe keine Palindromzahl ist, wird der Algorithmus erneut auf die entstandene Summe angewendet: **N** wird dabei durch **N+R** ersetzt.

Für viele Zahlen terminiert dieser Algorithmus nach wenigen Iterationen.
Beispiele:

```
N = 123 .... R = 321 .... Summe = 444 .... Palindrom,
fertig nach einer Iteration


N = 19  .... R = 91 .... Summe = 110 .... kein Palindrom,
N = 110 .... R = 11 .... Summe = 121 .... Palindrom,
fertig nach zwei Iterationen

N = 983  .... R = 389  .... Summe = 1372 .... kein Palindrom,
N = 1372 .... R = 2731 .... Summe = 4103 .... kein Palindrom,
N = 4103 .... R = 3014 .... Summe = 7117 Palindrom,
fertig nach drei Iterationen
```

Es gibt jedoch auch Zahlen, für die der Algorithmus (vermutlich) nie terminiert.

Implementieren Sie ein Java-Programm, welches beim Aufruf mit einer Dezimalzahl (z.B. 300) als Programmparameter (maximal 100000) alle Zahlen bis zu dieser Parameterzahl ausgibt, für die der o.a. Algorithmus **NICHT** (wie unten beschrieben) terminiert.

```
% java Nolindrom 300
196
295
```

soll demnach alle Zahlen mit der geforderten Eigenschaft bis 300 (eine pro Zeile) ausgeben. Es sind tatsächlich die beiden angegebenen. Falls das Programm ohne Argument aufgerufen wird, soll es eine entsprechende Fehlerausgabe erzeugen und sofort beendet werden.

```
% java Nolindrom
Bitte geben Sie die Obergrenze als Parameter an.
```

Sie können (vereinfachend) zunächst davon ausgehen, dass eine Zahl, die nach 100 Iterationen noch kein Palindrom erzeugt hat, vermutlich die geforderte Eigenschaft besitzt. Ob es tatsächlich (Dezimal-)Zahlen gibt, für die der Algorithmus **NIE** abbricht, ist übrigens völlig offen!

Für den direkten Nachweis, dass ein Algorithmus nicht terminiert, ist dessen Abarbeitung auf Rechnern nun mal leider nicht geeignet :-)

Da die entstehenden Zahlen in jedem Iterationsschritt größer werden (im Durchschnitt verdoppeln sie sich in jedem Schritt), haben wir allerdings mit der Zählung der Iterationsschritte (bis maximal 100) ein ganz anderes Problem: Wenn Sie für die Darstellung von **N** und **R** den Typ **`int`** verwenden, werden Sie feststellen, dass schon bei der Berechnung für **N=89** (stillschweigend) ein Überlauf stattfindet, der natürlich alle nachfolgenden Schritte falsch werden lässt. Eigentlich müsste für **N=89** nach 24 Iterationen das Palindrom **8813200023188** entstehen! Verwenden Sie also gleich den Typ **`long`** der allerdings auch von dem Problem des Überlaufs (nur etwas später) betroffen ist.

Benutzen Sie deshalb die Tatsache, dass u.U. bei der nächsten Addition **N+R** ein Überlauf stattfindet, direkt (**und ausschließlich!**) als Abbruchkriterium der oben beschriebenen Iteration (weit unter 100 Durchläufen). Informieren Sie sich dazu unter

https://docs.oracle.com/en/java/javase/11/docs/api/index.html [1]

in der Klasse **`Long`** über **`Long.MAX_VALUE`** und verwenden Sie diese Konstante im Programm (statt deren numerischen Wert 9223372036854775807).

Hilfestellung für Anfänger

Entwickeln (und testen) Sie Ihren Algorithmus stufenweise, so dass Sie sicher sind, dass der bisher erstellte (Teil-)Algorithmus korrekt arbeitet, bevor Sie ihn in der nächsten Stufe verwenden! Fangen Sie mit einem Kern des Algorithmus an, der zunächst für ein festes (aber im Test variierendes) **N** die Zahl **R** (korrekt!) bestimmt. Davon können Sie sich überzeugen, indem das Programm das errechnete **R** ausgibt:

```
…
public static void main(String[] s) {
      long N = 123; // und andere Werte, auch mal mit 0 als Einerziffer :-)
      // Ihr Kernalgorithmus …
      long R = … // der von Ihnen bestimmte ‚umgedrehte' Zahlenwert
      System.out.println("R = " + R);  // ist es 321 ?
}
…
```

Die nächste Stufe könnte sein, dass Sie den Überlauftest implementieren:

```
…
public static void main(String[] s) {
      long N = …; // Werte, die zum Ueberlauf fuehren oder nicht
      // Ihr Kernalgorithmus …
      long R = … // der von Ihnen bestimmte ‚umgedrehte' Zahlenwert
      // System.out.println("R = " + R); // wird nicht mehr gebraucht, kann
      // aber als Kommentar im Programm verbleiben und ggf. aktiviert werden
```

[1] oder auch in der Dokumentation einer anderen Java-Version

```
        if ( <N+R koennte zu einem Ueberlauf fuehren> )
                System.out.println("Ueberlauf");
        else
                System.out.println(N+R); // korrekt, ohne Ueberlauf
}
…
```

Danach vielleicht die Erkennung ob **N** schon ein Palindrom ist? Dafür haben Sie eigentlich schon alles parat:-)
Aufpassen: Ist die Startzahl selbst ein Palindrom (z.B. 4994) soll der Algorithmus noch nicht abbrechen, sondern wenigstens einmal **N+R** bestimmen …

Nun könnten Sie die Iteration einbauen, die in einer Schleife **N** durch **N+R** ersetzt, das neue **R** bestimmt und die Schleife verlassen, wenn ein Überlauf droht.

Abschließend (wenn bis dahin alles wie erwartet funktioniert) setzen Sie alles bisherige in eine Schleife, die **N** bis zur (als Programmparameter gegebene) Obergrenze variiert.

Zusatzaufgabe für Fortgeschrittene (5 Zusatzpunkte)

Die oben beschriebene Abbruchbedingung (bei drohendem Überlauf) führt u.U. dazu, dass Zahlen ausgegeben werden, die zu einem Palindrom geführt hätten (also genau genommen nicht zu den erwünschten Zahlen gehören), wenn wir die Berechnung nur lange genug (z.B. wirklich bis 100 Iterationen) hätten ausführen können. Ermitteln Sie (zusätzlich zur bislang beschriebenen Aufgabe und offenbar mit einer anderen Zahlenrepräsentation als **`long`**) die kleinste Zahl, bei der bei bis zu 100 Iterationen **doch** ein Palindrom entsteht, also die kleinste, die bei Überlauf-Abbruch in der Ausgabe erscheint, aber eigentlich nicht dazugehört (weil sie bei ‚exaktem' Rechnen in einem Palindrom endet)!

Ihr Programm soll dann (neben der oben beschriebenen Grundfunktionalität) einen zusätzlichen Parameter (**x** als 2. Argument) berücksichtigen:

```
% java Nolindrom 12345
196
295
…                                   wie gehabt
% java Nolindrom 1234 x
alle Zahlen werden auch durch Abbruch per Ueberlauf gefunden
% java Nolindrom 12345 x
????? braucht ?? Iterationen bis zum Palindrom ??? …… ???  [2]
```

ACHTUNG
Um die Korrektur der Aufgaben durch die Tutoren zu erleichtern, gelten wie immer (d.h. auch bei allen weiteren Aufgaben) die folgenden Regeln:

- Eingesandte Java-Programme, die nicht vom Java-(Referenz-)Compiler akzeptiert werden (d.h. Fehler bei der Übersetzung liefern) werden ohne weitere Inspektion mit 0 (Leistungs-) Punkten bewertet! Ob Sie die 5 Abgabepunkte erhalten, entscheiden wir in Abhängigkeit davon, ob Ihre Einsendung genügend mit der Aufgabe zu tun hat ;-)
- Halten Sie sich bei der Benennung von Klassen, Funktionen und Dateien immer an die Vorgaben und achten Sie darauf, dass Sie keine Packages verwenden (s.o.). Alle Abweichungen von diesen Regeln führen zu zusätzlichem Aufwand bei der Bewertung und daher auch zu Punktabzug.
- Die vom implementierten Programm erzeugten Ausgaben müssen das geforderte Format exakt einhalten (incl. Zeilenstruktur und Leerzeichen). Soll und Ist werden automatisch verglichen. Sofern Abweichungen festgestellt werden, gibt es Punktabzug, über dessen Höhe dann die Tutoren entscheiden! Die einzig zulässige Abweichung (eine die man bei der Textansicht des Programms und seiner Ausgaben u.U. gar nicht feststellen kann) besteht in der Codierung der Zeilenenden, die unter Windows mit zwei Zeichen (carriage return [/x0d] + line feed [/x0a]) unter Unix dagegen nur mit einem Zeichen (newline [/x0a]) dargestellt werden.

---

[2] statt der Fragezeichen soll Ihr Programm natürlich die richtige kleinste Zahl (wie oben beschrieben), die Anzahl der Iterationen für diese Zahl und das entstehende Palindrom ausgeben.