Aufgabe 3

Die Methode `pKR` berechnet die n-te Primzahl ($n \geq 1$) kaskadenartig rekursiv und äußerst ineffizient:

```java
  static long pKR(int n) {
    long p = 2;
    if (n >= 2) {
      p = pKR(n - 1); // beginne die Suche bei der vorhergehenden Primzahl
      int i = 0;
      do {
        p++; // pruefe, ob die jeweils naechste Zahl prim ist, d.h. ...
        for (i = 1; i < n && p % pKR(i) != 0; i++) {
        } // pruefe, ob unter den kleineren Primzahlen ein Teiler ist
      } while (i != n); // ... bis nur noch 1 und p Teiler von p sind
    }
    return p;
  }
```

github: raw

Überführen Sie `pKR` mittels *dynamischer Programmierung* (hier also *Memoization*) und mit möglichst *wenigen Änderungen* so in die *linear* rekursive Methode `pLR`, dass `pLR(n, new long[n + 1])` ebenfalls die n-te Primzahl ermittelt:

```java
private long pLR(int n, long[] ps) {
  ps[1] = 2;
  // ...
}
```

> **Exkurs: Kaskadenartig rekursiv**
> Kaskadenförmige Rekursion bezeichnet den Fall, in dem mehrere rekursive Aufrufe nebeneinander stehen.

> **Exkurs: Linear rekursiv**
> Die häufigste Rekursionsform ist die lineare Rekursion, bei der in jedem Fall der rekursiven Definition höchstens ein rekursiver Aufruf vorkommen darf.

```java
  static long pLR(int n, long[] ps) {
    ps[1] = 2;
    long p = 2;
    if (ps[n] != 0) return ps[n];
    if (n >= 2) {
      p = pLR(n - 1, ps); // beginne die Suche bei der vorhergehenden
       ↪ Primzahl
      int i = 0;
      do {
        p++; // pruefe, ob die jeweils naechste Zahl prim ist, d.h. ...
        for (i = 1; i < n && p % ps[i] != 0; i++) {
        } // pruefe, ob unter den kleineren Primzahlen ein Teiler ist
      } while (i != n); // ... bis nur noch 1 und p Teiler von p sind
    }
    ps[n] = p;
```

```
69      return p;
70    }
```

github: raw

Der komplette Quellcode

```
3   /**
4    * Berechne die n-te Primzahl.
5    *
6    * Eine Primzahl ist eine natürliche Zahl, die größer als 1 und
     ↪  ausschließlich
7    * durch sich selbst und durch 1 teilbar ist.
8    *
9    * <ul>
10   * <li>1. Primzahl: 2
11   * <li>2. Primzahl: 3
12   * <li>3. Primzahl: 5
13   * <li>4. Primzahl: 7
14   * <li>5. Primzahl: 11
15   * <li>6. Primzahl: 13
16   * <li>7. Primzahl: 17
17   * <li>8. Primzahl: 19
18   * <li>9. Primzahl: 23
19   * <li>10. Primzahl: 29
20   * </ul>
21   */
22  public class PrimzahlDP {
23
24    /**
25     * Die Methode pKR berechnet die n-te Primzahl (n >= 1) Kaskadenartig
     ↪  Rekursiv.
26     *
27     * @param n Die Nummer (n-te) der gesuchten Primzahl. Die Primzahl 2 ist
     ↪  die
28     *          erste Primzahl. Die Primzahl 3 ist die zweite Primzahl etc.
29     *
30     * @return Die gesuchte n-te Primzahl.
31     */
32    static long pKR(int n) {
33      long p = 2;
34      if (n >= 2) {
35        p = pKR(n - 1); // beginne die Suche bei der vorhergehenden Primzahl
36        int i = 0;
37        do {
38          p++; // pruefe, ob die jeweils naechste Zahl prim ist, d.h. ...
39          for (i = 1; i < n && p % pKR(i) != 0; i++) {
40          } // pruefe, ob unter den kleineren Primzahlen ein Teiler ist
41        } while (i != n); // ... bis nur noch 1 und p Teiler von p sind
42      }
43      return p;
44    }
45
46    /**
47     * Die Methode pLR berechnet die n-te Primzahl (n >= 1) Linear Rekursiv.
48     *
49     * @param n Die Nummer (n-te) der gesuchten Primzahl. Die Primzahl 2 ist
     ↪  die
50     *          erste Primzahl. Die Primzahl 3 ist die zweite Primzahl etc.
51     * @param ps Primzahl Speicher. Muss mit n + 1 initialisert werden.
```

```
52      *
53      * @return Die gesuchte n-te Primzahl.
54      */
55     static long pLR(int n, long[] ps) {
56       ps[1] = 2;
57       long p = 2;
58       if (ps[n] != 0) return ps[n];
59       if (n >= 2) {
60         p = pLR(n - 1, ps); // beginne die Suche bei der vorhergehenden
           ↪ Primzahl
61         int i = 0;
62         do {
63           p++; // pruefe, ob die jeweils naechste Zahl prim ist, d.h. ...
64           for (i = 1; i < n && p % ps[i] != 0; i++) {
65           } // pruefe, ob unter den kleineren Primzahlen ein Teiler ist
66         } while (i != n); // ... bis nur noch 1 und p Teiler von p sind
67       }
68       ps[n] = p;
69       return p;
70     }
71
72     static void debug(int n) {
73
        ↪ System.out.println(String.format("%d. Primzahl: %d (kaskadenartig rekursiv berechnet)",
        ↪ n, pKR(n)));
74       System.out
75
            ↪ .println(String.format("%d. Primzahl: %d (linear rekursiv berechnet)",
            ↪ n, pLR(n, new long[n + 1])));
76     }
77
78     public static void main(String[] args) {
79       System.out.println(pKR(10));
80       System.out.println(pLR(10, new long[11]));
81
82       for (int i = 1; i <= 10; i++) {
83         debug(i);
84       }
85     }
86   }
```

github: raw