Sieb des Erathostenes

Mit Hilfe des klassischen Algorithmus "Sieb des Erathostenes" (ca. 300 v.Chr.) sollen alle Primzahlen (i.e. eine ganze Zahl, die nur durch sich selbst und durch 1 teilbar ist) bis zu einer einzugebenden Obergrenze bestimmt werden.

Da das mit Divisionen sehr aufwändig ist, hatte Erathostenes folgende Idee: Man schreibt die Zahlen von 1 bis zur Obergrenze auf. Dann streicht man daraus die Vielfachen von 2, 3, 5 usw. bis zur Quadratwurzel aus der Obergrenze. Die nicht gestrichenen Zahlen sind am Ende dann die Primzahlen.

Entwickle einen Algorithmus und ein Java-Programm, das diese Funktion automatisiert. Achte dabei auch auf Header, Kommentare, so wie eine übersichtlich strukturierte Form.

Ein gewissenhafter Test samt Dokumentation sollte ebenfalls gemacht werden.

1.) Überlege welche Testfälle (für Benutzereingaben) probiert werden sollten.

2.) Schreibe diese Testfälle als Liste nieder.

3.) Führe die Tests durch und dokumentiere, welche zu Fehler führten bzw. funktionieren.

4.) Ändere gegebenenfalls dein Programm und wiederhole die Tests samt Dokumentation.

Hinweis: Dazu verwenden wir einen boolschen Array `isPrim` mit `n` Elementen den wir mit `true` initialisieren. Ist ein beliebiges Element `isPrim[i]` `false`, dann wurde es bereits gestrichen und ist somit sicher keine Primzahl; ist es `true`, dann KANN es sich dabei um eine Primzahl handeln, so ferne sie nicht später noch gestrichen wird.

Sonderaufgabe für Spezialisten: Zähle die Anzahl der Zugriffe auf den Array `isPrim` und Versuche, diese zu minimieren! Hintergrund: in der Kryptografie werden sehr große Primzahlen zur Verschlüsselung benötigt. Die größten Computer rechnen Wochen und Monate um solche Zahlen zu finden – da ist ein höchst effizienter Algorithmus natürlich Pflicht!

Ausgabe:

```
Obergrenze: 100

Primzahlen von 1..100: 1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47,
53, 59, 61, 67, 71, 73, 79, 83, 89, 97

Anz. der Primzahlen von 1..100 : 26
```