1. Gyakorlat

Java ismétlés

1. Feladat - Prímszámok keresése

Írj Java programot, ami megkeresi egy adott n számnál kisebb prímszámokat! Készíts többféle algoritmust a kereséshez.

1. Naív algoritmus: ellenőriz minden számot 2 és n között, hogy prímek-e. A k szám vizsgálata abból áll, hogy megnézi, hogy 2 és $\sqrt{k}$ között van-e osztója.
2. Eratoszthenész szitája. Lásd: Wikipedia
3. Az előző két algoritmus egyfajta kombinációja: ellenőriz minden számot 2 és n között, hogy prímek-e, de egy k szám vizsgálata most abból áll, hogy csak a korábban megtalált prímekkel teszteli az oszthatóságot. (Itt is elég nézni az oszthatóságot az eddig megtalált azon prímekkel, amelyek nem nagyobbak $\sqrt{k}$-nál.)

Törekedj objektumorientált megoldásra! Pl.: `FindPrimes` interfész static factory metódusokkal, melyeken keresztül az implementációk elérhetők.

A `System.currentTimeMillis()` és `System.nanoTime()` segítségével mérd meg a futási időt. Végezz több mérést és készíts statisztikát az eredményekből (a kilógó értékeket el lehet hagyni).

Házi feladat

Fejezd be az órán elkezdett feladatot.

Szorgalmi feladat - mátrixszorzás

Az $A \in \mathbb{R}^{m \times n}$ és a $B \in \mathbb{R}^{n \times p}$ mátrixok szorzata egy $C \in \mathbb{R}^{m \times p}$ mátrix, melynek elemei a következők:

$$c_{i,j} = \sum_{k=1}^{n} a_{ik} b_{k_j}, \quad i = 1, \ldots, m, \quad j = 1, \ldots, p$$

1. Implementáld a mátrixszorzást Java tömbökön!
2. Mérd meg a futási időt!
3. Vizsgáld meg, milyen hatással van a belső for-ciklusok felcserélése a futási időre!